NGA.SIG.0026.05_1.0_ACCSPEC
2017-05-30

NGA STANDARDIZATION DOCUMENT

Accuracy and Predicted Accuracy in the NSG: Specification and Validation

Technical Guidance Document (TGD) 2c

(2017-05-30)

Version 1.0

Forward

This handbook is approved for use by all Departments and Agencies of the Department of Defense.

Comments, suggestions, or questions on this document should be addressed to the GWG World Geodetic System (WGS) and Geomatics (WGSG) Focus Group, ATTN : Chair, WGS/Geomatics Standards Focus Group, ncgis-mail.nga.mil or to the National Geospatial-Intelligence Agency Office of Geomatics (SFN), Mail Stop L-41, 3838 Vogel Road, Arnold, MO 63010 or emailed to GandG@nga.mil.

Summary of Changes and Modifications

Revision	Date	Status	Description
1.0	20161117	FINAL	INITIAL PUBLICATION

Contents

1 Scope

This Technical Guidance Document (TGD 2c) is a specific topic document on specifications for and validation of geospatial accuracy in technology, system and product acquisition, part of a series of information and guidance documents regarding Accuracy and Predicted Accuracy in the National System for Geospatial Intelligence (NSG). As the title suggests, it focuses on the need, methods, and practices to adequately specify and communicate the geospatial accuracy requirements for an acquisition activity and the validation processes to evaluate satisfaction of the demonstrated performance against those specifications within the context of a larger scope of work which includes a more generalized overview and additional topic specific technical guidance. Documents in this series are listed below:

TGD 1	Accuracy and Predicted Accuracy in the NSG:	Overview and Methodologies
TGD 2a	Accuracy and Predicted Accuracy in the NSG:	Predictive Statistics
TGD 2b	Accuracy and Predicted Accuracy in the NSG:	Sample Statistics
TGD 2c	Accuracy and Predicted Accuracy in the NSG:	Specification and Validation
TGD 2d	Accuracy and Predicted Accuracy in the NSG:	Estimators and Quality Control
TGD 2e	Accuracy and Predicted Accuracy in the NSG:	Monte-Carlo Simulation
TGD 2f	Accuracy and Predicted Accuracy in the NSG:	External Data and Quality Assessment

The series is also supported by a compiled glossary of relevant terms:

TGD 1-G	Accuracy and Predicted Accuracy in the NSG:	Glossary of Terms

All documents in the series, “Accuracy and Predicted Accuracy in the NSG”, are intended to provide technical guidance to inform the development of geospatial data accuracy characterization for NSG GEOINT collectors, producers and consumers -- accuracy characterization as required to describe the trustworthiness of geolocations for defense and intelligence use and to support practices that acquire, generate, process, exploit, and provide geolocation data and information based on geolocation data. Today, both the sources and desired uses for geospatial data are quickly expanding. Throughout the NSG, trusted conveyance of geospatial accuracy is broadly required for a variety of traditional and evolving missions including those supported by manual, man-in-the-loop, and automated processes. This guidance is the foundation layer for a collection of common techniques, methods, and algorithms ensuring that geospatial data within the NSG can be clearly requested, delivered and evaluated as fit for desired purpose whether by decision makers, intelligence analysts, or as input to further processing techniques.

TGD 2c contains references to and is referenced by other Technical Guidance Documents. The documents in this series, TGD 1, TGD 2a - TGD 2b, and TGD 2d - TGD 2f, also have cross-references

among themselves. All Technical Guidance Documents also reference external public as well as "NGA approved for public release" documents for further insight/details.

The TGD 2 documents, including this document focused on specification and validation, are also considered somewhat top-level in that they are not directed at specific systems. They do provide general guidance, technical insight, and recommended algorithms. The relationship of the Technical Guidance Documents with specific GEOINT Standards documents and specific Program Requirements documents is presented in Figure 1-1, where arrows refer to references. That is, in general, specific product requirement documents reference specific GEOINT standards documents which reference specific technical guidance documents.

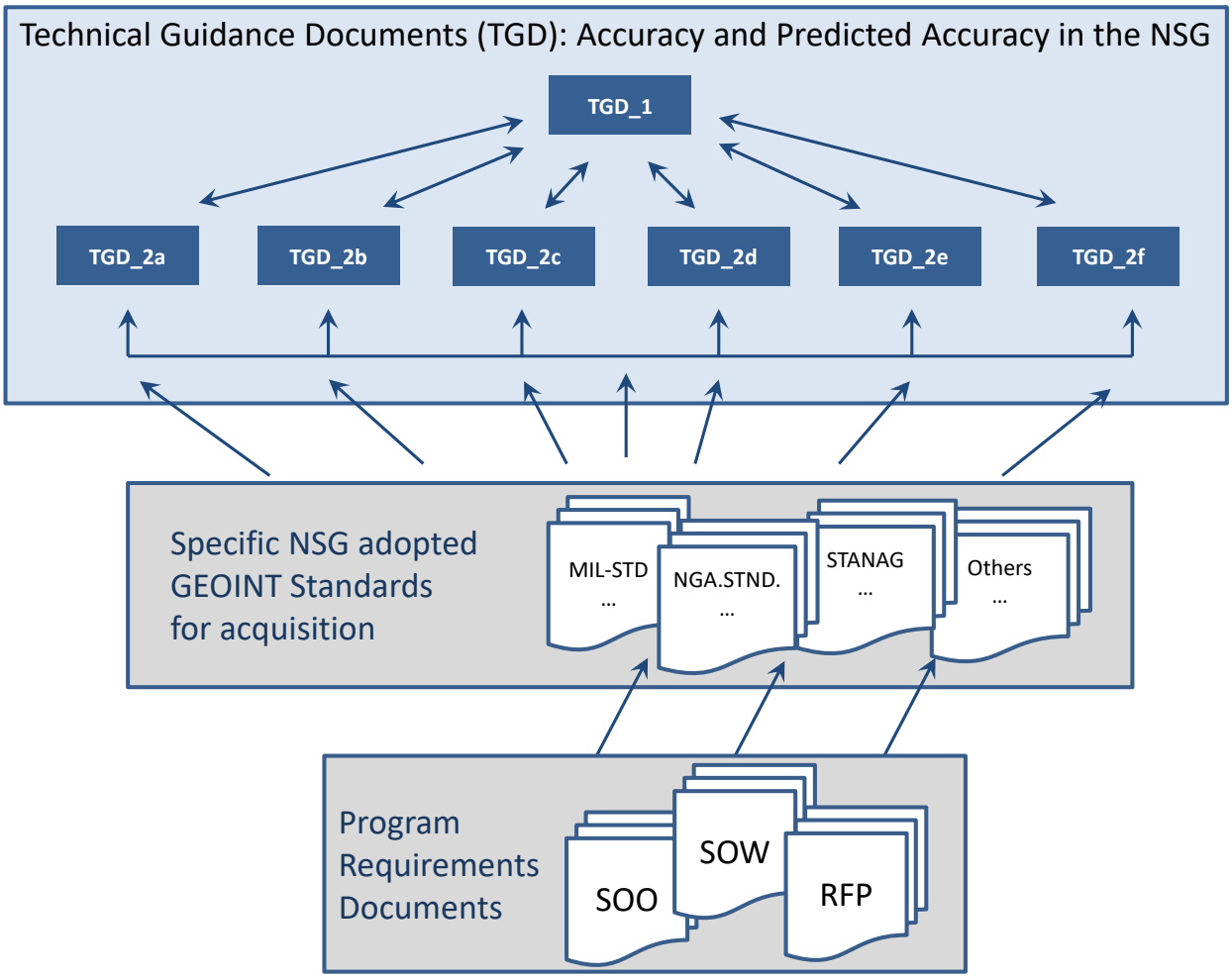


Figure 1-1: The relationships between the Technical Guidance Documents, GEOINT Standards Documents, and Program Requirement Documents

Accuracy and Predicted Accuracy in the NSG: Specification and Validation, Technical Guidance Document (TGD) 2c is for guidance only and cannot be cited as a requirement.

2 Applicable Documents

The documents listed below are not necessarily all of the documents referenced herein, but are those needed to understand the information provided by this information and guidance document.

2.1 Government specifications, standards, and handbooks

NGA.SIG.0026.01_1.0_ACCOVER, Accuracy and Predicted Accuracy in the NSG: Overview and Methodologies, Technical Guidance Document (TGD) 1

NGA.SIG.0026.02_1.0_ACCGLOS, Accuracy and Predicted Accuracy in the NSG: Glossary of Terms, Technical Guidance Document (TGD) 1-G

NGA.SIG.0026.03_1.0_ACCPRED, Accuracy and Predicted Accuracy in the NSG: Predictive Statistics, Technical Guidance Document (TGD) 2a

NGA.SIG.0026.04_1.0_ACCSAMP, Accuracy and Predicted Accuracy in the NSG: Sample Statistics, Technical Guidance Document (TGD) 2b

NGA.SIG.0026.06_1.0_ACCESQC, Accuracy and Predicted Accuracy in the NSG: Estimators and Quality Control, Technical Guidance Document (TGD) 2d

NGA.SIG.0026.07_1.0_ACCMTCO, Accuracy and Predicted Accuracy in the NSG: Monte-Carlo Simulation, Technical Guidance Document (TGD) 2e

NGA.SIG.0026.08_1.0_ACCXDQA, Accuracy and Predicted Accuracy in the NSG: External Data and Quality Assessment, Technical Guidance Document (TGD) 2f

3 Definitions

There are a number of authoritative guides as well as existing standards within the NSG and Department of Defense for definitions of the identified key terms used in this technical guidance document. In many cases, the existing definitions provided by these sources are either too general or, in some cases, too narrow or dated by intended purposes contemporary to the document's development and publication. The definitions provided in this document have been expanded and refined to explicitly address details relevant to the current and desired future use of accuracy in the NSG. To acknowledge the basis and/or lineage of certain terms in Section 3.1, we reference the following sources considered as either foundational or contributory:

[a] Anderson, James M. and Mikhail, E., Surveying: Theory and Practice, 7th Edition, WCB/McGraw-Hill, 1998.

[b] DMA-TR-8400.1, DMA Technical Report: Error Theory as Applied to Mapping, Charting, and Geodesy.

[c] Defense Mapping Agency, Glossary of Mapping, Charting, and Geodetic Terms, 4th Edition, Defense Mapping Agency Hydrographic/Topographic Center, 1981.

[d] ISO TC/211 211n2047, Text for ISO 19111 Geographic Information - Spatial referencing by coordinates, as sent to the ISO Central Secretariat for issuing as FDIS, July 17, 2006.

[e] Joint Publication (JP) 1-02, Department of Defense Dictionary of Military and Associated Terms, November 8, 2010 as amended through January 15, 2016.

[f] MIL-HDBK-850, Military Handbook: Glossary of Mapping, Charting, and Geodetic Terms, January 21, 1994.

[g] MIL-STD-2401, Department of Defense Standard Practice; Department of Defense World Geodetic System (WGS), January 11, 1994

[h] MIL-STD-600001, Department of Defense Standard Practice; Mapping, Charting and Geodesy Accuracy, February 26, 1990.

[i] *National System for Geospatial Intelligence* [Brochure] Public Release Case #15-489.

[j] NGA.STND.0046_1.0, The Generic Point-cloud Model (GPM): Implementation and Exploitation, Version 1.0, October 03, 2015.

[k] Oxford Dictionaries (www.oxforddictionaries.com/us/) copyright © 2016 by Oxford University Press.

[l] Soler, Tomas and Hothem, L., "Coordinate Systems Used in Geodesy: Basic Definitions and Concepts", Journal of Surveying Engineering, Vol. 114, No. 2, May 1988.

3.1 Key Terms Used in the Document

3.1.1 Accuracy

The range of values for the error in an object's metric value with respect to an accepted reference value expressed as a probability. [f]

- Statements of accuracy may be developed through applications of predictive statistics or by sample statistics based on multiple independent samples of errors.

3.1.2 Circular Error

See "Section 3.1.11 Scalar Accuracy Metrics".

3.1.3 Error

The difference between the observed or estimated value and its ideal or true value. See Appendix A for a more detailed and augmented definition. [f]

3.1.4 Ground Truth

The reference or (assumed) true value of a geolocation of a measured quantity (e.g. associated with an absolute geolocation, or a relative mensuration).

3.1.5 Linear Error

See "Section 3.1.11 Scalar Accuracy Metrics".

3.1.6 Monte-Carlo Simulation

A technique in which a large number of independent sample inputs for a system are randomly generated using an assumed *a priori* statistical model to analyze corresponding system output samples statistically and support derivation of a statistical model of the system output. This technique is valuable for complex systems, non-linear systems, and those where no insight to internal algorithms is provided ("black box" systems).

3.1.7 National System for Geospatial Intelligence (NSG)

The operating framework supported by producers, consumers or influencers of geospatial intelligence (GEOINT). Spanning defense, intelligence, civil, commercial, academic and international sectors, the NSG contributes to the overall advancement of the GEOINT function within the strategic priorities identified by the Functional Manager for Geospatial Intelligence in the role established by Executive Order 12333. The framework facilitates community strategy, policy, governance, standards and requirements to ensure responsive, integrated national security capabilities. [i]

3.1.8 Predicted Accuracy

The range of values for the error in a specific object's metric value expressed as a probability derived from an underlying and accompanying detailed statistical error model.

- If the statistical error model does not include the identification of a specific probability distribution, a Gaussian (or Normal) probability distribution is typically assumed in order to generate probabilities.
- The term "Predicted" in Predicted Accuracy corresponds to the use of predictive statistics in the detailed statistical error model; it does not correspond to a prediction of accuracy applicable to the future since the corresponding error corresponds to a geolocation already extracted.

3.1.9 Predictive Statistics

Statistics corresponding to the mathematical modeling of assumed *a priori* error characteristics contained in a statistical error model.

3.1.10 Sample Statistics

Statistics corresponding to the analysis of a collection of physical observations, a sample of the population, as compared to an assumed true or an *a priori* value.

3.1.11 Scalar Accuracy Metrics

Convenient one-number summaries of geolocation accuracy and geolocation predicted accuracy expressed as a probability: (1) Linear Error (LE) corresponds to 90% probable vertical error, (2) Circular Error (CE) correspond to 90% probable horizontal radial error, and (3) Spherical Error (SE) corresponds to 90% spherical radial error. [b],[f],and [h] See Appendix A for a more detailed and augmented definition.

3.1.12 Statistical Error Model

Information which describes the error data corresponding to a given state vector. The information includes the type of corresponding error representation (random variable, random vector, stochastic process, or random process), the category of statistics (predictive or sample), and associated statistical information including at a minimum the mean-value and covariance data.

3.1.13 Validation

The process of determining the degree to which a model is an accurate representation of the real world from the perspective of its intended use/s. In the NSG, this includes validation of accuracy and predicted accuracy specified capabilities. [e]

3.2 Other Relevant Terms

Appendix A contains definitions of the following additional terms relevant to the content of this document:

- *A priori*
- *A posteriori*
- Absolute Horizontal Accuracy
- Absolute Vertical Accuracy
- Bias Error
- Confidence Ellipsoid
- Confidence Interval
- Confidence Interval (Order Statistics)
- Correlated Error
- Correlated Values
- Covariance
- Covariance Matrix
- Cross-covariance Matrix
- Error (augmented definition)
- Error Ellipsoid
- Estimator
- Gaussian (or Normal) probability distribution
- Horizontal Error
- Inter-state vector correlation
- Intra-state vector correlation
- Least-upper-bound (lub)
- Local Tangent Plane Coordinate System
- Mean-Value
- Multi-Image Geopositioning (MIG)
- Order Statistics
- Percentile
- Precision
- Probability density function (pdf)
- Probability distribution
- Probability distribution function (cdf)
- Radial Error
- Random Error
- Random Error Vector
- Random Variable
- Random Vector
- Realization
- Relative Horizontal Accuracy
- Relative Vertical Accuracy
- Rigorous Error Propagation
- Scalar Accuracy Metrics (augmented definition)
- Standard Deviation
- State Vector
- State Vector Error
- Uncertainty
- Uncorrelated Error
- Uncorrelated Values
- Variance
- Vertical Error

3.3 Abbreviations and Acronyms

Abbreviation/Acronym	Definition
1d	One Dimensional
2d	Two Dimensional
3d	Three Dimensional
CE	Circular Error
ENU	East North Up
EO	Electro-optical
GWG	Geospatial Intelligence Standards Working Group
i.i.d.	independent and identically distributed
LE	Linear Error
lub	least-upper-bound
MIG	Multi-Image Geopositioning
NSG	National System for Geospatial Intelligence
rmse	root-mean-square error
SE	Spherical Error
TGD	Technical Guidance Document
WGS	World Geodetic System
WGSG	World Geodetic System and Geomatics

4 Introduction to Specification and Validation of Accuracy and Predicted Accuracy in the NSG

This document presents detailed technical guidance regarding the specification and validation of accuracy requirements as well as predicted accuracy requirements in the NSG. The default accuracy addressed is absolute accuracy, although relative accuracy is considered as well. For a little more context, the definitions of (absolute) accuracy and (absolute) predicted accuracy are as follows:

Accuracy in the NSG is defined as: "the range of values for the error in an object's metric value expressed as a probability". Furthermore, this general definition can be sub-allocated to more specific accuracies. For example, we can define horizontal accuracy for a specific system as: "the 90th percentile of horizontal (radial) geolocation error, where location is relative to a specified geodetic reference system".

Predicted accuracy in the NSG is defined as: "the range of values for the error in a specific object's metric value expressed as a probability derived from an underlying and accompanying detailed statistical error model." The detailed statistical error model includes predictive statistics, with the key statistic being the predicted error covariance matrix relative to the error in the object's metric value – typically the 3d error in a specific extracted geolocation.

A top-level discussion of accuracy and predicted accuracy in the NSG is provided in TGD 1: "Accuracy and Predicted Accuracy in the NSG: Overview and Methodologies". Both accuracy and predicted accuracy are critical to overall operational performance.

In addition, the proper operational representation, use, and metric values associated with both accuracy and predicted accuracy in an NSG system extend to the system's specification of performance requirements and their validation. Without proper specification and validation, performance cannot be planned, developed, demonstrated, or maintained. In general, accuracy and predicted accuracy refer to 3d geolocation error, the error in the "end product" of an NSG geolocation system.

TGD 1 presents an overview of this specification and validation process for a typical NSG geolocation system consisting of three generic top-level modules: Collection, Value-added Processing, and Exploitation, as summarized in Figure 4-1 below. The X_i and $C_{\epsilon X_i}$ in the figure correspond to independent samples of 3d geolocations and corresponding $3x3$ predicted error covariance matrices, respectively, typically generated by an (near) optimal extraction process that necessarily outputs both. (For example, a Weighted Least Squares unbiased estimator with rigorous error propagation – see Section 5.8.1 of TGD 1.) For additional details on this overall top-level specification and validation process, see Section 5.1 of TGD 1.

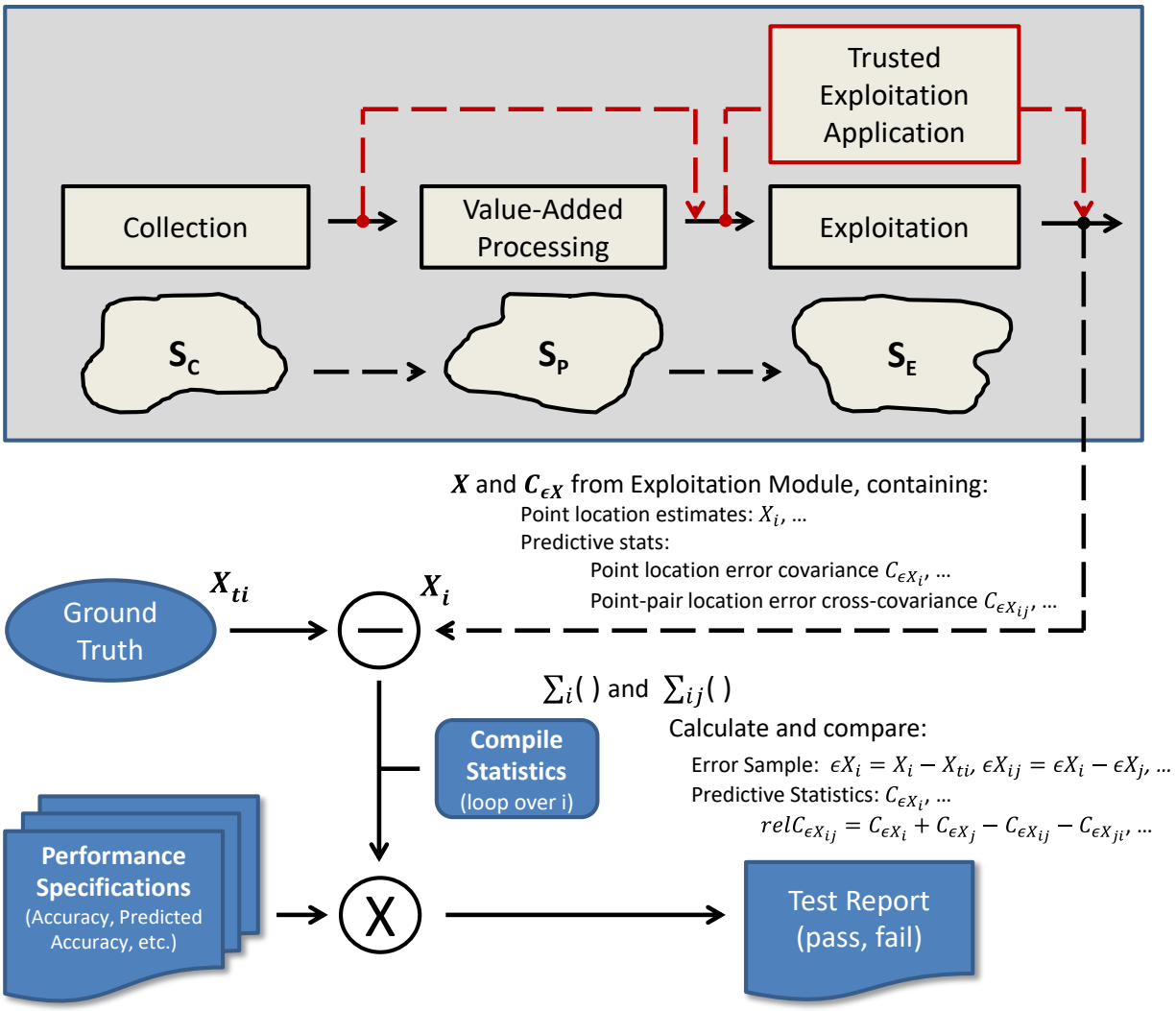


Figure 4-1: Specification and Validation of Accuracy and Predicted Accuracy for an NSG Geolocation System

This document concentrates on the recommended procedures, algorithms, and equations associated with the lower half (below the gray box) of the above figure. Both accuracy and predicted accuracy are assessed. Both assessments rely on independent error samples ϵX_i generated by differencing the geolocations X_i from available "ground truth" (accurate surveyed locations). In addition, the assessment of predicted accuracy relies on the geolocations' corresponding predicted error covariance matrices $C_{\epsilon X_i}$ in order to generate normalized error samples at various levels of probability.

This document also examines and addresses the potential for Type I and Type II errors in validation of an NSG Geolocation System. In general, Type I validation errors refer to validation of a system that should pass but incorrectly fails and Type II validation errors refer to validation of a system that should fail but incorrectly passes. The specific character and methods for mitigating Type I and Type II errors in

validation of accuracy and of predicted accuracy differ and are discussed individually within the respective sections.

Sections 4.1 and 4.2 now go on to present a more detailed overview of the above specification and validation processes. Section 4.3 then provides a detailed technical guide to the contents of Section 5, which presents specific details of the specification and validation processes. These processes also support the following top-level principles implemented throughout this document:

- Specification and validation of predicted accuracy should be performed in addition to the specification and validation of accuracy
- The specifications of accuracy and predicted accuracy (performance) should also take into consideration and address their corresponding validations
- Appropriate i.i.d. (independent and identically distributed) error samples should be used in the validation processes
- Validation processes should be practical and realistic – not just based on theory
- Validation processes should include confidence metrics

Finally, the following summarizes top-level definitions/symbology for various geolocation errors used throughout the remainder of the document. The error components ϵx, ϵy, and ϵz are assumed to correspond to a local tangent plane coordinate system, such as East-North-Up (ENU), the symbol superscript "T" corresponds to "vector transpose", and the symbol "≡" corresponds to "defined as":

- Vertical error $\epsilon X \equiv [\epsilon z]$ Vertical radial error $\epsilon v \equiv \sqrt{\epsilon z^2}$ (4-1)
- Horizontal error $\epsilon X \equiv [\epsilon x \quad \epsilon y]^T$ Horizontal radial error $\epsilon h \equiv \sqrt{\epsilon x^2 + \epsilon y^2}$
- 3d error $\epsilon X \equiv [\epsilon x \quad \epsilon y \quad \epsilon z]^T$ 3d radial error $\epsilon r \equiv \sqrt{\epsilon x^2 + \epsilon y^2 + \epsilon z^2}$

4.1 Overview of Specification & Validation of Accuracy Requirements

The specification and validation of accuracy of an NSG geolocation system is straightforward and involves requirements represented using the ubiquitous scalar accuracy metrics LE_XX, CE_XX, and SE_XX, which are equivalent to the XX percentile of vertical, horizontal, and 3d radial errors, respectively, where XX = 50, 90, or 95%. Corresponding radial error samples are used for validation of the accuracy requirement.

The following overview addresses horizontal accuracy for specificity, and in particular, the recommended form for its specification and the recommended procedure for its validation. However, as detailed in Section 5.1, any combination of vertical, horizontal, and 3d accuracy can be specified and validated in a similar manner.

The specification of horizontal accuracy for an NSG geolocation system is as follows:

Specification of horizontal accuracy requirements

Horizontal geolocations shall satisfy the following:

$\epsilon h_{XX} \leq CEXX_{spec}$, where (4.1-1)

$\epsilon h \equiv \sqrt{\epsilon x^2 + \epsilon y^2}$ is defined as horizontal radial error (always positive unlike the horizontal error components ϵx and ϵy),

ϵh_{XX} is the XX percentile of the random variable ϵh, i.e., $prob\{\epsilon h \leq \epsilon h_{XX}\} = 0.XX$, and

$CEXX_{spec}$ is the specified value for horizontal circular error at the XX% probability (percentile) level, where XX=50%, 90%, or 95% , with 90% the default.

Note: ϵh_{XX} is also termed the XX percentile of horizontal radial error.

Note: $CEXX$ is defined identical to the definition of ϵh_{XX}; the use of $CEXX_{spec}$ in Equation (4.1-1) for the specified requirement instead of an equivalent ϵh_{XXspec} is due primarily to legacy convention.

The above specification can also be written in the equivalent form (at the default probability or percentile level $XX = 90$ in the following example):

$prob\{\epsilon h \leq CE90_{spec}\} \geq 0.90.$ (4.1-2)

Thus, it is at least 90% probable that horizontal radial error for an arbitrary extraction is less than the specified value $CE90_{spec}$.

The specified value $CE90_{spec}$ is NSG-system specific. For example, for an NSG geolocation system based on commercial satellite imagery, a possible value for $CE90_{spec}$ is 5.5 meters.

The use of root-mean-square error (rmse) instead of scalar accuracy metrics is not recommended for the specification and validation of accuracy as explained later in this section. If need be, rmse can be approximately converted to equivalent scalar accuracy metrics.

The specification of accuracy requirements includes a top-level description of the assumed operational scenario (constraints) for geolocation extraction, e.g., imaging angles for commercial satellite imagery – see Section 5 for more details.

Validation of horizontal accuracy requirements

The recommended process for the validation of the above specification (Equation (4.1-1)) is based on order statistics which are applicable to an arbitrary (unknown) probability distribution of the underlying (ϵx and ϵy) errors – a mean-value of zero and/or a Gaussian probability distribution are not required – a desirable and robust feature.

In particular, given a set of i.i.d. horizontal radial error samples $\{\epsilon h_k, k = 1, .., n\}$, the (probabilistic) least-upper-bound (lub) of the percentile ϵh_{XX} is computed: $lub\_\epsilon h_{XX}$. Furthermore, it is computed at a specified (minimum) level of confidence (or probability) YY (%).

Thus, by definition of lub:

$$prob\{\epsilon h_{XX} < lub\_\epsilon h_{XX}\} \geq 0.YY, \qquad (4.1\text{-}3)$$

with the default confidence-level YY equal to 90% (see Appendix A of this document for a formal definition of lub and TGD 2b, Section 5.3.6 for more detailed discussion).

In particular, during validation the least-upper-bound $lub\_\epsilon h_{XX}$ is set equal to the smallest ordered sample value (ordered by ascending magnitude) of horizontal radial error such that Equation (4.1-3) is satisfied. Thus, we are at least 90% confident that the true (and unknown) value of ϵh_{XX} is less than the computed value $lub\_\epsilon h_{XX}$. The percentile level and the confidence level can be specified independently, both with default values of 90%. And finally, if the computed $lub\_\epsilon h_{XX} \leq CE_{spec}$, validation is successful. On the other hand, if the computed $lub\_\epsilon h_{XX} > CE_{spec}$, validation fails.

The use of a least-upper-bound (one-sided confidence interval) is critical for the validation of accuracy, as the NSG must have confidence in its results.

(Note: as detailed later in Section 5.1, the least-upper-bound can be computed for either vertical, horizontal, or 3d radial errors, computed for either 50, 90, or 95% (XX) percentiles, and computed at either 50, 90, or 95% confidence levels (YY)).

Validation of accuracy requirements is "plot friendly"

The above validation process is simple, straight-forward, and "plot-friendly" for additional insight and confidence in the validation results. This is illustrated in Figure 4.1-1 corresponding to $n = 100$ i.i.d. error samples. The blue circles correspond to the samples ϵh_k of horizontal radial error, the magenta line the value $lub\_\epsilon h_{90}$ computed from these samples, and the dotted red-line the best estimate of ϵh_{90} computed from these samples as well for ancillary information. The best estimate of ϵh_{90} corresponds to the value of the 90th ordered sample and the least-upper-bound $lub\_\epsilon h_{90}$ corresponds to the value of the 95th ordered sample (samples ordered by ascending magnitude, not as shown in Figure 4.1-1). Note that selection of the 90th ordered sample for the best estimate of ϵh_{90} corresponds to (0.90 percentile x 100 order samples) = 90. The least-upper-bound $lub\_\epsilon h_{90}$ contains an additional "pad" (higher order sample number) that is required in order to ensure, at a 90% confidence, that the true but unknown ϵh_{90} is less than or equal to the computed least-upper-bound.

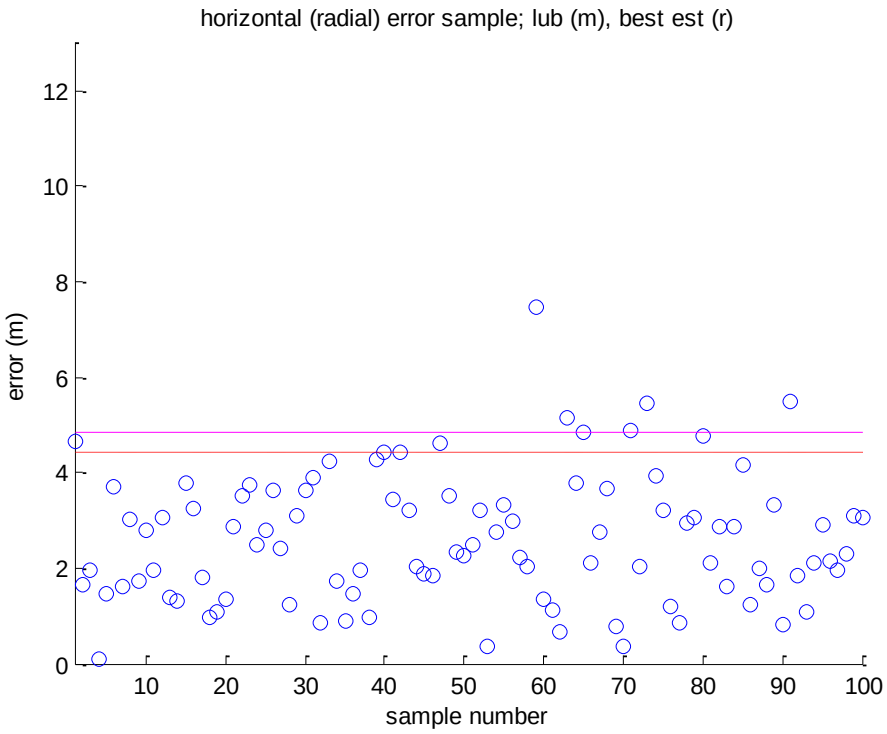


Figure 4.1-1: Example of the successful Validation of Horizontal Accuracy based on 100 i.i.d. samples of horizontal radial error; solid magenta line the value of $lub\_\epsilon h_{90}$, dotted red line the value of the best estimate of ϵh_{90}

Note that validation was successful for this particular example, as $lub\_\epsilon h_{90} = 4.9$ meters is less than the specified $CE90_{spec} = 5.5$ meters. That is, we are at least 90% confident that the true and unknown value of the 90th percentile ϵh_{90} is less than the computed value $lub\_\epsilon h_{90}$ which is less than the specified requirement. The best estimate of the 90th percentile ϵh_{90} was also computed for ancillary information, is independent of confidence-level, and is equal to 4.4 meters in this example. And, of course, being an estimate, the best estimate is not actually equal to the true value of the 90th percentile ϵh_{90}. That is why $lub\_\epsilon h_{90}$ is used for validation instead – it has a "built-in" measure of confidence in its value relative to the true but unknown value ϵh_{90}. And as proven in Section 5.3 of TGD2b, this computed confidence is theoretically rigorous – rather remarkable given that there is no assumption regarding the probability distribution of the underlying errors.

As a reminder, 90% is the default value for the percentile-level of interest for horizontal radial error, and 90% is the default value for the confidence-level of interest for computation of the corresponding least-upper-bound. However, as detailed in Section 5.1, both the percentile and the confidence-level can be specified independently from the values 50%, 90%, and 95%, if so desired. This is illustrated in Table 4.1-1 applicable to the 100 horizontal radial error samples of Figure 4.1-1 above. The sensitivity of table entries to both the specified percentile level and the specified confidence-level also provides general insight into the properties of order statistics.

Table 4.1-1: The best estimate (best est) and the least-upper-bound (lub) of the XX percentile of horizontal radial error based on 100 i.i.d. samples (Figure 4.1-1) and the use of order statistics

percentile (%):	best est / lub order sample #			best est / lub value (m)		
	confidence level (%):			confidence level (%):		
	50	90	95	50	90	95
50	50 / 51	50 / 57	50 / 59	2.5 / 2.55	2.5 / 2.7	2.5 / 2.8
90	90 / 91	90 / 95	90 / 96	4.4 / 4.45	4.4 / 4.9	4.4 / 5.0
95	95 / 96	95 / 99	95 / 100	4.9 / 5.0	4.9 / 5.4	4.9 / 7.7

The best estimate of the XX percentile of horizontal radial error ϵh_{XX} is a function of the specified percentile but independent of the specified confidence-level. On the other hand, the least-upper-bound is a function of both. In general, for a given percentile XX and a given number of i.i.d. samples, the higher the confidence-level YY specified for the least-upper-bound, the larger the corresponding order sample number and larger the corresponding value. As can be seen from the above table, for the default percentile value 90% and for the default least-upper-bound confidence-level value 90%, order sample numbers 90 and 95, with corresponding values of 4.4 and 4.9 meters, are applicable for the best estimate and the least-upper-bound of the (true) horizontal radial error percentile ϵh_{90}, respectively.

As discussed in Section 4.1.3 of this document, pseudo-code is supplied that performs the entire accuracy validation process, including the appropriate computations for the best estimate and the least-upper-bound of the horizontal radial error percentile ϵh_{XX} based on order statistics that were featured in the above example. (For those interested in underling computational details, see TGD 2b, Sections 5.3 and 5.4.)

The above example also assumed the use of 100 i.i.d. error samples. The recommended number of error samples for the validation of specified accuracy requirements is discussed in the next section, Section 4.1.1. But first, an issue associated with the possible use of root-mean-square error (rmse) instead of scalar accuracy metrics (e.g. $CEXX_{spec}$) for the specification of accuracy is addressed:

Root-mean-square error (rmse) is neither specified nor validated; it can be converted when necessary

As described earlier, it is recommended that a horizontal geolocation accuracy requirement be specified in terms of the scalar accuracy metric $CEXX_{spec}$. For NSG applications, it is essential that a geolocation accuracy requirement corresponds to a specifiable probability level XX, which this specification does. The verification of this accuracy requirement is also performed using order statistics, which have the desirable feature that a specific probability distribution for the underling errors is neither required nor used. In summary, the above recommended approach takes into account the probability of error but does not require a specific probability distribution.

Sometimes an NSG geolocation system will make use of (integrate) a sensor whose performance is specified by its manufacturer/operator in terms of root-mean-square error (rmse), not in terms of the preferred scalar accuracy metric $CEXX$. Rmse is a statistical parameter that has the desirable characteristic of independence from a specific probability distribution, but the unfortunate characteristic of not being associated with a specific level of probability.

Rmse is not recommended for the specification of geolocation accuracy in the NSG nor is the computation of its sample-statistic recommended for the validation of accuracy. Various reasons are discussed in Section I.1 of Appendix I. However, the "bottom line" is this: For most NSG applications, it is essential to know that it is 90% probable that the horizontal (radial) error of an extraction is less than a specified value $CE90_{spec}$, not that its expected (average) magnitude is less than a specified value $rmse_{spec}$. The latter tells us virtually nothing about the likelihood of larger errors unless we are also willing to assume a specific probability distribution of errors, which is an approximation at best. (Note: more correctly, rmse corresponds to the square-root of horizontal (radial) error's expected magnitude-squared – virtually the same thing as the expected magnitude in most cases.)

Section I.1 also presents approximate conversion factors for the conversion of rmse to $CEXX$ for XX equal to 50, 90, and 95%, to be used if necessary. If an rmse is provided, it is recommended that it be converted to $CEXX$ and the latter subsequently used as the baseline value for $CEXX_{spec}$. The recommended format for the NSG specification of accuracy and its validation based on order statistics that were presented earlier are then applicable and should be followed.

The remainder of this document, other than Section I.1, no longer considers rmse, i.e., the recommended approach for the specification and validation of geolocation accuracy is as described earlier. (Note: Section I.1 also converts rmse for vertical radial errors to $LEXX$ and rmse for 3d radial errors to $SEXX$, for use in the specification of vertical and 3d accuracies and their validation, respectively.)

4.1.1 Recommended Number of Error Samples

Validation of horizontal accuracy requirements is based on the computation of the least-upper-bound of the XX percentile of corresponding radial errors. It is recommended that the least-upper-bound (lub) be computed at a $YY = 90\%$ confidence-level with at least 100 i.i.d. samples. As few as 40 samples can also be used with certain restrictive caveats as explained below. However, 40 samples is the firm minimum for the number of i.i.d. samples. (These same recommendations are applicable to the validation of vertical and 3d accuracy requirements as well.)

Type II validation errors correspond to the event that validation should not pass but does. The 90% confidence-level for the lub ensures that there is less than a 10% probability of a Type II validation error. Type II validation errors are also essentially independent of the number of i.i.d. error samples.

Type I validation errors correspond to the event that validation should pass but does not. The probability of Type I validation errors is both a function of the number of samples and the geolocation system's "design margin": the difference between the required or specified accuracy $CE90_{spec}$ and the actual (but unknown) accuracy, or more specifically, the 90th percentile of horizontal radial error ϵh_{90}. The larger a positive design margin ($CE90_{spec} - \epsilon h_{90} > 0$) and the larger the number of samples used in validation, the lower the probability of a Type I validation error. An "intermediate" quantity, termed a "validation pad", is related and defined as the difference between the lub and ϵh_{90}.

The validation pad provides insight into the overall issue of Type I validation errors and is discussed in detail below. This discussion is then followed by details of Type I and Type II validation errors as a function of design margin and parameterized by the number of samples. The discussion includes easy-to-read plots of the probability of Type I validation errors versus design margin and the probability of Type II validation errors versus design margin.

Finally, formal validation of accuracy is possible with fewer than 100 i.i.d. error samples, but is more difficult unless there is a large design margin. As few as 40 i.i.d. samples can be used with the understanding that the probability of a Type I validation error will be significantly larger for most reasonable design margins. (See Section 4.1.2 for discussion of exceptions related to low accuracy systems.) And it must be emphasized that these are to be i.i.d. samples, not correlated samples – see both Section 5.2.6 of TGD 2b and Section 5.6 of this document for relevant discussions regarding i.i.d. error samples. The above recommendations corresponding to the number of samples are also applicable to all radial error percentiles, i.e., XX = 50, 90, and 95%.

Validation pad

The following provides insight into why at least 100 i.i.d. samples are recommended for validation. As was indicated in Figure 4.1-1 and Table 4.1-1 corresponding to the previous example, the least-upper-bound $lub\_\epsilon h_{90}$ at a $YY = 90\%$ confidence-level is larger than the best estimate of ϵh_{90}, where ϵh_{90} corresponds to the true (unknown) value of the $XX = 90^{\text{th}}$ percentile of horizontal radial error (aka true CE_90). Furthermore, $lub\_\epsilon h_{90}$ is larger than ϵh_{90} itself with a confidence of $YY = 90\%$ by definition.

The difference between $lub\_\epsilon h_{90}$ and ϵh_{90} constitutes a $YY = 90\%$ confidence-level or "**validation pad**". This pad is defined as $pad \equiv (lub\_\epsilon h_{90} - \epsilon h_{90})$, and is positive-valued $YY = 90\%$ of the time. If the pad becomes too big, it is less likely that the specified requirement $CE90_{spec}$ will be validated, i.e., less likely that $lub\_\epsilon h_{90} \leq CE90_{spec}$. This pad is a function of the number of samples used to compute the least-upper-bound – the smaller the number of samples the larger the pad. Related quantities are illustrated conceptually in Figure 4.1.1-1 below.

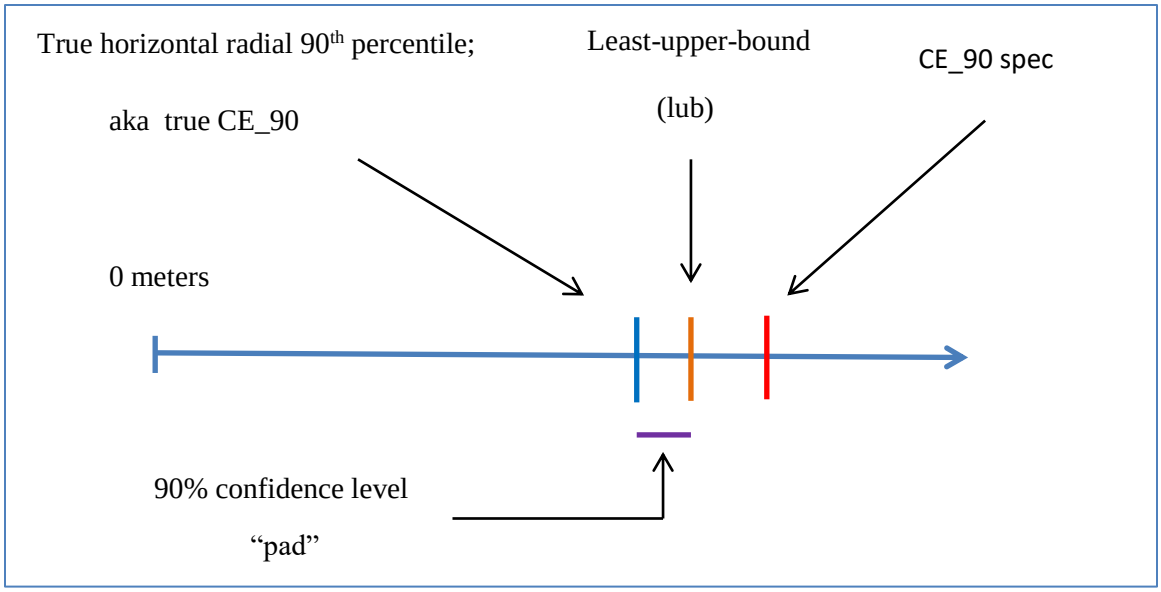


Figure 4.1.1-1: Conceptual illustration of a 90% confidence-level or validation pad

rel_pad is defined as the value of the pad relative to the true value of ϵh_{90} (meters), i.e., $rel\_pad \equiv pad/\epsilon h_{90} = (lub\_\epsilon h_{90} - \epsilon h_{90})/\epsilon h_{90}$. The blue curve in the following Figure 4.1.1-2 presents the expected (average) value of rel_pad in percent as a function of the number of samples used.

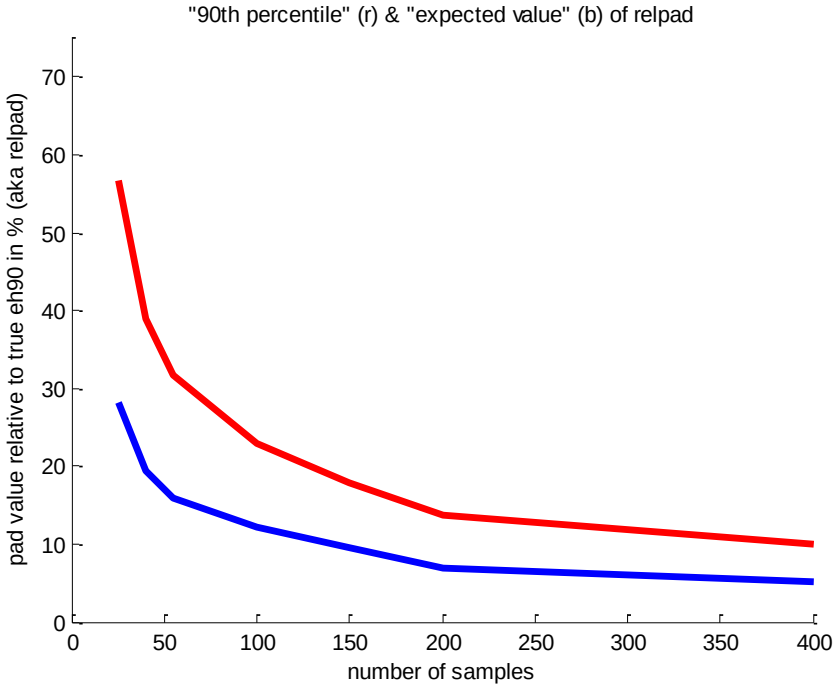


Figure 4.1.1-2: Expected value (blue) and $ZZ = 90^{th}$ percentile (red) of the validation pad relative to the true value ϵh_{90} (or rel_pad) in percent

In particular, if only 25 i.i.d. samples are available, the value of rel_pad is expected to be approximately 28%. If 100 i.i.d. samples are available instead, the value of rel_pad is expected to be only 13% instead. And as seen in Figure 4.1.1-2, there is also a substantial "knee in the curve" at around 40 to 100 i.i.d. error samples. The value of rel_pad changes most dramatically as a function of the number of samples during this range of samples. Thus, for example, adding 50 more i.i.d. samples to 40 i.i.d. samples has a much more dramatic effect on rel_pad than adding 50 more i.i.d. samples to 150 i.i.d. samples.

As stated above, the blue curve in Figure 4.1.1-2 corresponds to the expected or average value of rel_pad in percent. It illustrates the effect of the number of i.i.d. samples on the value or "length" of the (relative) validation pad. However, a $ZZ = 90^{\text{th}}$ percentile value of rel_pad is a more realistic (and larger) value as demonstrated in the next paragraph, and is termed max_rel_pad for convenience. It corresponds to the red curve in the above figure with results summarized as follows as a function of the number of i.i.d. error samples n_{samp}:

- $max\_rel\_pad = 10\%$ if $n_{samp} = 400$ (4.1.1-1)
- $max\_rel\_pad = 14\%$ if $n_{samp} = 200$
- $max\_rel\_pad = 23\%$ if $n_{samp} = 100$
- $max\_rel\_pad = 33\%$ if $n_{samp} = 55$
- $max\_rel\_pad = 39\%$ if $n_{samp} = 40$
- $max\_rel\_pad = 56\%$ if $n_{samp} = 25$

If $CE90_{spec} = (1 + max\_rel\_pad) \times \epsilon h_{90}$ there is an approximate $ZZ = 90\%$ probability that validation correctly passes. Of course, the value of max_rel_pad depends on the number of i.i.d. error samples; if 100 samples, $max\_rel\_pad = 0.23$, and $CE90_{spec}$ would have to equal $1.23 \times \epsilon h_{90}$ in order for validation to correctly pass with a probability of 90%. And assuming the same number of i.i.d. error samples, if $CE90_{spec}$ is even larger, the probability that validation correctly passes is larger as well. Also, if $CE90_{spec} = (1 + expected\_value\_rel\_pad) \times \epsilon h_{90}$ instead, there would only be an approximate 50% probability that validation correctly passes.

The more general situation in which $CE90_{spec} \equiv (1 + \Delta) \times \epsilon h_{90}$ corresponds to an accuracy requirement based on a "**design margin**", with the corresponding design margin equal to Δ. The term "design margin" refers to a general design/analytic process used by an NSG geolocation system in order to determine the required value $CE90_{spec}$. If $\Delta = max\_rel\_pad$, there is a 90% probability that validation correctly passes by definition.

The term "**estimated design margin**" is used for Δ when the value for ϵh_{90} in the defining equation $CE90_{spec} \equiv (1 + \Delta) \times \epsilon h_{90}$ is an *a priori* estimate from the design process as well, as opposed to its true value. The general concept and utility of a design margin and an estimated design margin are addressed under the next sub-heading and in the following subsection, respectively.

Appendix H describes the generation of Figure 4.1.1-2 as well as the results of Equation (4.1.1-1) based on a Monte-Carlo simulation [TGD2e] implemented via (included) pseudo-code. Simulation results are

also somewhat sensitive to the ratio of the eigenvalues of the corresponding *a priori* (assumed) 2×2 covariance matrix of horizontal errors; however, the values of Equation (4.1.1-1) are reasonably representative as discussed in the appendix.

Type I and Type II Errors in accuracy validation

Type I validation errors are defined as the event that validation of accuracy should pass but does not: $\epsilon h_{90} \leq CE90_{spec}$ but $lub\_\epsilon h_{90} > CE90_{spec}$.

Type II validation errors are defined as the event that validation of accuracy should not pass but does: $\epsilon h_{90} > CE90_{spec}$ but $lub\_\epsilon h_{90} \leq CE90_{spec}$.

Figure 4.1.1-3 presents the probability of passing validation as a function of the design margin Δ, where $CE90_{spec} = (1 + \Delta) \times \epsilon h_{90}$. Results are further parameterized by the number of i.i.d. samples (25, 40, 55, 100, 200).

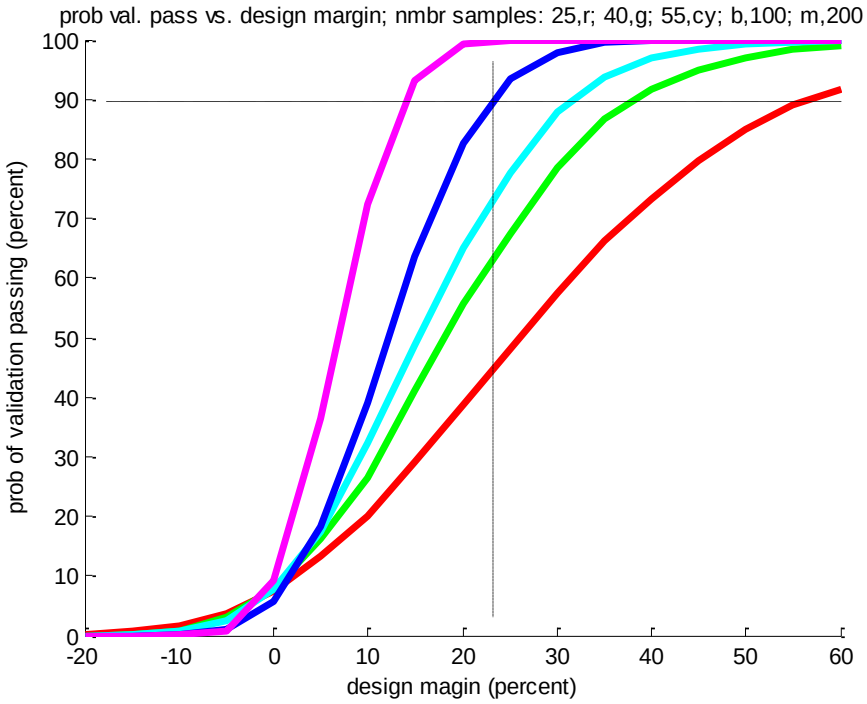


Figure 4.1.1-3: Probability of passing validation as a function of the design margin Δ in percent; parameterized by the number of i.i.d. error samples

The value $\Delta = 0.23$ in the above figure equals max_rel_pad corresponding to 100 samples.

Figure 4.1.1-4 corresponds to the probability of Type I validation errors, and Figure 4.1.1-5 corresponds to the probability of Type II validation errors. Type I validation errors only occur when Δ is positive, and Type II validation errors only occur when Δ is negative. The results of Figures 4.1.1-3 through 4.1.1-5 are based on Monte Carlo simulation, with corresponding pseudo-code presented in Appendix H.

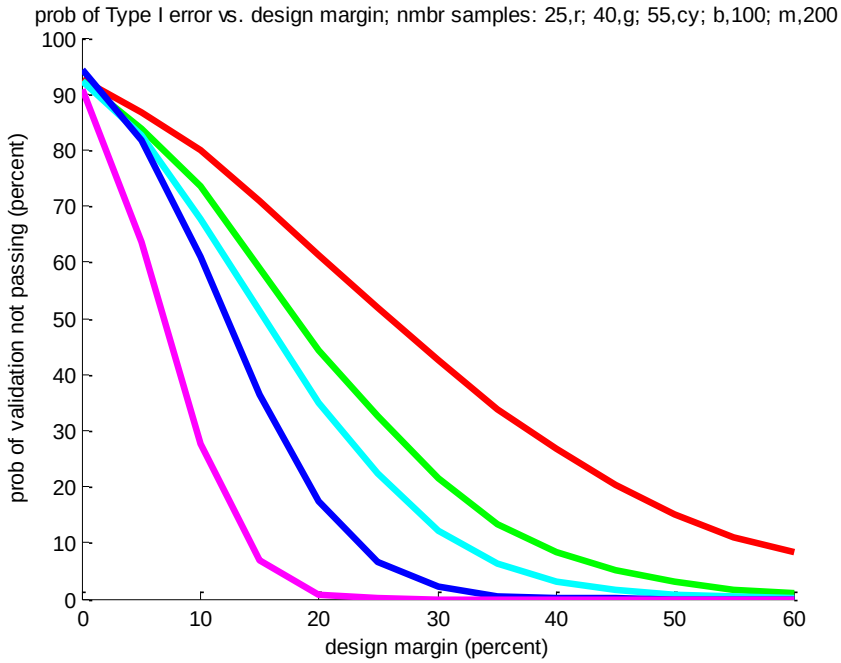


Figure 4.1.1-4: Probability of Type I validation errors as a function of the design margin Δ in percent; parameterized by the number of i.i.d. error samples

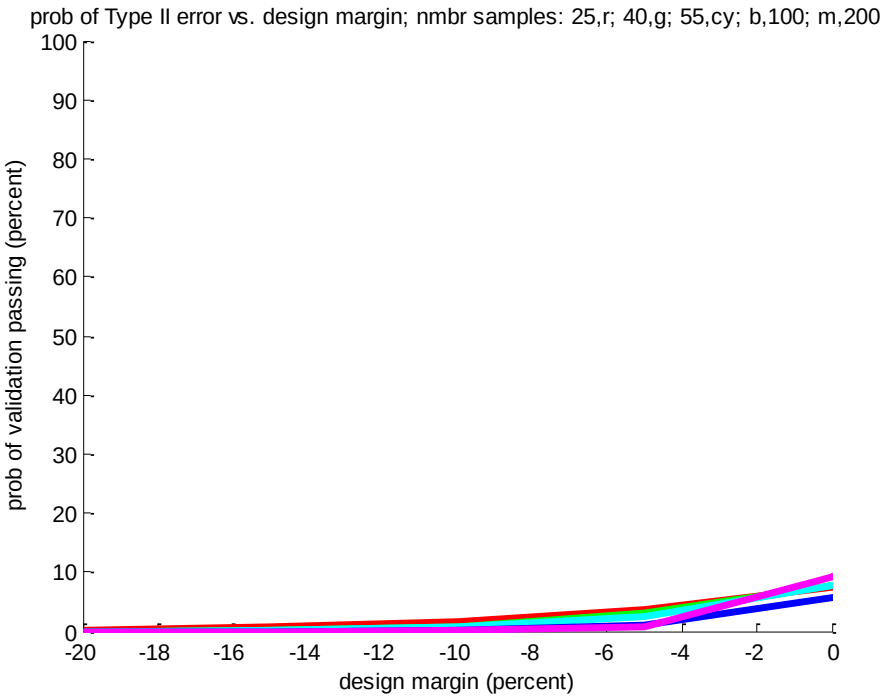


Figure 4.1.1-5: Probability of Type II validation errors as a function of the design margin Δ in percent; parameterized by the number of i.i.d. error samples

Type I validation errors are minimized through the use of more i.i.d. samples, which implies a smaller validation pad per Equation (4.1.1-1). That is, $lub\_\epsilon h_{90}$ is closer to the true ϵh_{90} and hence it is more likely to correctly pass validation. This is why at least a 100 i.i.d. samples are recommended for validation.

The minimization of Type II validation errors is a "given" via the use of an lub with a corresponding reasonably large confidence-level. This is why a $YY = 90\%$ confidence-level is recommended for validation. And as seen by comparing Figures 4.1.1-4 and 4.1.1-5, the recommended approach to validation emphasizes the minimization of Type II validation errors over the minimization of Type I validation errors. That is, if a validation error does occur, it is far more preferable that it correspond to validation failing when it should have passed (Type I error) than it correspond to validation passing when it should have failed (Type II error).

In addition to the use of more i.i.d. samples, Type I validation errors are also minimized via the amount of (positive) design margin Δ, if any, built into the geolocation system, as discussed further in the next subsection.

4.1.2 The Bigger Picture for Specification

Due to the above "validation pad" in conjunction with a normal system design margin, the prudent practice is that a viable NSG geolocation system accuracy specification will have at least a $\Delta = 25\%$ overall and built-in estimated design margin relative to validation of the actual system's operational performance, i.e., $CE90_{spec} \geq (1 + 0.25)\epsilon h_{90}$, where the value used for ϵh_{90} is an *a priori* estimate. Of course, there is an interrelationship between the top-level specified accuracy requirement(s), the (prime) contractor's system (and subsystem) design requirements which flow down from the specified requirements, and the validation of the actual system's performance.

For example, suppose that an NSG geolocation system to be based on commercial satellite imagery has a true but hypothetical NSG mission requirement of $CE90_{mission} = 5$ meters with required performance set equal, i.e., $CE90_{spec} = 5$ meters. It is assumed that the contractor/integrator will design to a performance level of 4 meters or better, i.e., $5 = (1 + 0.25) \times 4$ meters. This, of course, is not a requirement on the contractor/integrator per se – just good sense. On the other hand, it is recommended that the NSG only levy the 5 meter requirement if it feels that at least 4 meters is achievable with current technology and also consistent with the amount of funding allocated to the contractor/integrator in order to build the system or enhance/integrate a current system. In addition, the contractor/integrator's 4 meter allocated design requirement should be consistent with its estimates of the effects of validation, preferably based on a good estimate of the number of i.i.d. error samples to be available to validation. It is recommended that a minimum value for the number of i.i.d. error samples available to validation be included in the actual system accuracy requirement statement to which the contractor/integrator bids and designs.

The above relies on an adequate and built-in estimated design margin in order to successfully accommodate the effects or "length" of the corresponding validation pad in the validation of $CE90_{spec}$. An alternate, but non-preferred method is to directly add the validation pad to $CE90_{spec}$. For example,

based on an assumed number of i.i.d. samples available to validation equal to 55, max_rel_pad equals approximately 33% (see Equation (4.1.1-1)). This value is relative to the true value of ϵh_{90} which is further assumed to equal $CE90_{spec}$ for this alternate method, i.e., corresponds to an estimated design margin equal to zero ($\Delta = 0$). Therefore, an adjusted specification, to be used for validation only, is as follows: $adj\_CE90_{spec} = (1 + 0.33) \times CE90_{spec} = 1.33 \times CE90_{spec}$, where the value for the estimated design margin is now equal to $\Delta = 33\%$. The problem with this method is that the original value of zero for the estimated design margin was simply an assumption, not necessarily a reasonable estimate, and a negative value may actually be applicable instead, i.e., $CE90_{spec}$ may actually be less than the true value of ϵh_{90}. Consequently, there is a higher probability of a Type II validation error relative to the original specification when using the adjusted specification during validation.

An exception regarding the number of samples for low-accuracy products

In this document, the specification and validation of accuracy is primarily aimed at a geolocation system and its inherent geolocation capabilities. In some cases, such a system may also (or only) generate products that are based on its inherent geolocation capabilities but the products have their own and significantly less stringent accuracy requirements. For example, an NSG geolocation system may acquire and use registered (controlled) monoscopic near-nadir images generated by a particular commercial satellite imaging system that has an estimated geolocation accuracy of $CE_{est}^{sensor} = 10$ meters for corresponding horizontal extraction accuracy. The NSG geolocation system generates 1:50,000 maps based on these images. The map has a specified horizontal accuracy requirement of only approximately $CE_{spec}^{map} = 50$ meters. (Vertical accuracy is ignored in this example for simplicity.)

There are many independent errors that affect the geolocation accuracy of the end-product or map, such as: (1) the (image) mensuration error associated with the classification, identification, and extraction of an appropriate feature of interest for the map, (2) the error associated with the interpolation of geolocations between extracted and adjacent feature nodes, (3) the error associated with the map's scale and resolution, and (4) the inherent geolocation error associated with the geolocation system itself – that is, the error in the extraction of an arbitrary horizontal geolocation assuming only one-pixel mensuration error and none of the other identified map-specific errors.

In terms of the total map generation error budget of $CE_{spec}^{map} = 50$ meters, inherent geolocation error (accuracy) is allocated by the NSG geolocation system in this example to a value of $CE_{allocated}^{sensor} = 25$ meters. The combined effects of the other errors are allocated $CE_{allocated}^{other} = 43$, such that the root-sum-square of $CE_{allocated}^{sensor}$ and $CE_{allocated}^{other}$ equals $CE_{spec}^{map} = 50$ meters. The value $CE_{allocated}^{sensor} = 25$ meters corresponds to inherent geolocation accuracy and is to be validated.

(Validation of the overall accuracy of the map may also be applicable, but this is a different process, involves comparing various locations read from the map itself to corresponding "ground truth" locations, and is not addressed explicitly in this document.)

Setting the *a priori* estimate of ϵh_sensor_{90} to CE_{est}^{sensor}, and setting $CE_{spec}^{sensor} = CE_{allocated}^{sensor}$, it follows that $CE_{spec}^{sensor} = (1 + \Delta)CE_{est}^{sensor}$, or $CE_{spec}^{sensor} = (1 + 1.5)10 = 25$ meters with a corresponding 150%

estimated design margin. Based on earlier analysis (Equation (4.1.1-1)), there is a 56% validation pad (max_rel_pad) required relative to the true value ϵh_sensor_{90} using 25 i.i.d. error samples. There is also a corresponding 90% probability of correctly passing validation using a design margin equal to this validation pad, which is significantly less than the 150% estimated design margin that is available. Consequently, as few as 25 samples can be used for the formal validation of $CE_{allocated}^{sensor}$ if such samples are difficult to obtain. However, regardless the value of the estimated design margin, it is recommended that no fewer than 25 i.i.d. samples ever be used for formal validation.

For this particular example, and following the validation process for the specified requirement of $CE_{spec}^{sensor} = 25$ meters using 25 i.i.d. samples, it is expected that validation will pass easily with a value of $lub\_\epsilon h_{90}$ on the order of 13 meters (less than 16 meters at 90% probability), and a value of the best estimate of ϵh_{90}, computed for ancillary information only, on the order of 10 meters. The values 10, 13, and 16 meters corresponds to $CE_{est}^{sensor} = 10$ meters, $(1 + \exp\_value\_rel\_pad) \times CE_{est}^{sensor} = 1.28 \times CE_{est}^{sensor} \cong 13$ meters, and $(1 + max\_rel\_pad) \times CE_{est}^{sensor} = 1.56 \times CE_{est}^{sensor} \cong 16$ meters, respectively.

4.1.3 Pseudo-code for the Validation of Accuracy Requirements

Appendix B contains pseudo-code (MATLAB) that performs the entire accuracy validation process given the appropriate inputs per accompanying documentation and in compliance with the above Section 4.1 overview and the corresponding details presented in Section 5.1. Pseudo-code output includes plots similar to Figure 4.1-1 presented earlier. Appendix B also contains examples of the pseudo-code's explicit use.

4.2 Overview of Specification & Validation of Predicted Accuracy Requirements

The specification and validation of predicted accuracy is more complicated than the specification and validation of accuracy. It necessarily involves various probability levels, and both the use of geolocation (error) samples as well as their corresponding predicted error covariance matrices. The latter are used to normalize radial error samples at different levels of probability. Corresponding and appropriate normalization results validate that the covariance matrices are reliable, which in turn, ensure reliable predicted accuracies for the geolocations – reliable predicted scalar accuracy metrics can be computed from the error covariance matrix and geolocations can be optimally fused with other geospatial "products" using the (full) error covariance matrix directly [TGD1].

Validation of predicted accuracy implies validation of error models

As discussed briefly in Sections 4 and 5.1, a predicted error covariance matrix corresponds to the geolocation extraction process for a particular target or geospatial object, and is typically computed simultaneously with a best estimate of the geolocation. For the validation process, this covariance matrix also corresponds to a particular geolocation error: extracted geolocation differenced from corresponding ground truth location. The validation process is to ensure that this covariance matrix (predicted accuracy) is reliable, i.e., statistically reflects actual geolocation errors. In order for the covariance matrix to be reliable and validation of predicted accuracy to pass, the extraction process

must have performed rigorous error propagation and implemented reliable (predictive) statistical error models for all significant errors affecting the extracted geolocation, such as sensor metadata (sensor position, etc.) errors. (See TGD 1, Sections 5.2 and 5.8 for details regarding statistical error models and rigorous error propagation.)

Specification and validation of predicted accuracy require an assumed probability distribution

The specification and validation of predicted accuracy necessarily assume a specific form for the probability distribution of the underlying error components – an approximate mean-zero, multi-variate Gaussian distribution was selected. Thus, order statistics, which makes no assumption regarding probability distribution, and its (probabilistic) least-upper-bound are not applicable and not used.

More specifically, and regarding the assumed probability distribution for horizontal errors, $\epsilon X = [\epsilon x \quad \epsilon y]^T$ is assumed to have a mean-zero, multi-variate Gaussian (Normal) probability distribution, but with an arbitrary (valid) $2x2$ predicted covariance matrix. The covariance matrix can have non-identical diagonal elements and have off-diagonal elements as well. (Correspondingly, the scalar random variable horizontal radial error $\epsilon h = \sqrt{\epsilon x^2 + \epsilon y^2}$ has a distribution more general than either a Rayleigh or a Rice distribution.)

As discussed in Sections 5.2 and 5.4, validation also accounts for the fact that ϵX is only approximately mean-zero, multi-variate Gaussian distributed and is reasonably robust to this approximation. However, a multi-variate Gaussian probability distribution is still an assumption, and future research regarding the effects of this assumption as well as possible modifications given additional *a priori* information about the distribution is recommended. This is discussed in Section 5.4.4.

Now that the above background has been supplied, specification of predicted accuracy requirements is directly addressed followed by details of their validation. In particular, the specification of predicted horizontal accuracy is as follows (again, any combination of predicted vertical, horizontal, and 3d accuracy can actually be specified as detailed in Section 5.2):

<u>Specification of predicted horizontal accuracy requirements</u>

Normalized horizontal error shall satisfy the following:

- $prob\{\epsilon h\_norm_{99} \leq 1\} \geq 0.YY_{h\_99\_spec}$ and (4.2-1)
- $prob\{\epsilon h\_norm_{90} \leq 1\} \geq 0.YY_{h\_90\_spec}$ and
- $prob\{\epsilon h\_norm_{50} > 1\} \geq 0.YY_{h\_50\_spec}$, where

    - ***Normalized horizontal error tolerance requirements*** at the 99, 90, and 50% probability levels are specified as the three values $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$:
        - In order to provide context, typical values for a well-calibrated commercial satellite EO imaging system and numerous samples of error are on the order of $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$ = $\{97,\ 85,\ 44\ \%\}$.

- A more general discussion on where appropriate values are obtained is provided in a later paragraph.

- $\epsilon h\_norm_{XX} \equiv \epsilon h / \epsilon h\_pred\_radial_{XX}$
  - ϵh is the horizontal radial error for an arbitrary extraction
  - $\epsilon h\_pred\_radial_{XX}$ is the corresponding predicted radial (expected magnitude) of ϵh at the probability-level XX and computed from the predicted error covariance matrix
    - For example, there is a 90% probability that the horizontal radial error ϵh is less than or equal to $\epsilon h\_pred\_radial_{90}$
  - ϵh_norm_{XX} is also termed normalized horizontal radial error at the XX level of probability
  - $prob$ in Equation (4.2-1) is probability taken over all applicable extractions

The specification of predicted accuracy requirements includes a top-level description of the assumed operational scenario (constraints) for geolocation extraction, e.g., imaging angles for commercial satellite imagery – see Section 5 for more details.

Validation of predicted horizontal accuracy requirements

Validation of the above requirements (Equation (4.2-1)) simply consists of testing whether all of the specific normalized error tests (at the three different levels of probability) are successful, where probabilities ("$prob$") are computed as the percentage of normalized error samples passing the associated normalized error test, and where normalized error samples are computed as:

- $\epsilon h\_norm_{XX\_s\_k} = \epsilon h_{s\_k} / (d_{XX} \epsilon h_{s\_k} (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{-1/2})$ for an arbitrary horizontal error sample k, and where the numerator ϵh_{s_k} is the (scalar) horizontal radial error sample, and the denominator is the corresponding predicted radial at the XX (99, 90, or 50%) probability-level. (4.2-2)

  The denominator or predicted radial is computed as detailed above using the horizontal radial error sample ϵh_{s_k} , the $2x1$ horizontal (not radial) error sample ϵX_{s_k}, the corresponding $2x2$ predicted error covariance matrix $C_X\_pred_{s\_k}$, and assumes an (approximate) multi-variate Gaussian probability distribution of underlying horizontal error components ϵx and ϵy. This computation is also termed "ellipsoidal-based" since scalar accuracy metrics (e.g. $CE90$) are not used – more on this later. (Note that $d_{99} = 3.035$, $d_{90} = 2.146$, $d_{50} = 1.177$ (unit-less) corresponding to a multivariate (bivariate) Gaussian distribution.)

The horizontal radial error sample and its corresponding predicted radial at the 90% probability-level are represented graphically in the following figure. The 90% error ellipse is centered at zero and it is 90% probable that an arbitrary horizontal error is within its boundary by definition.

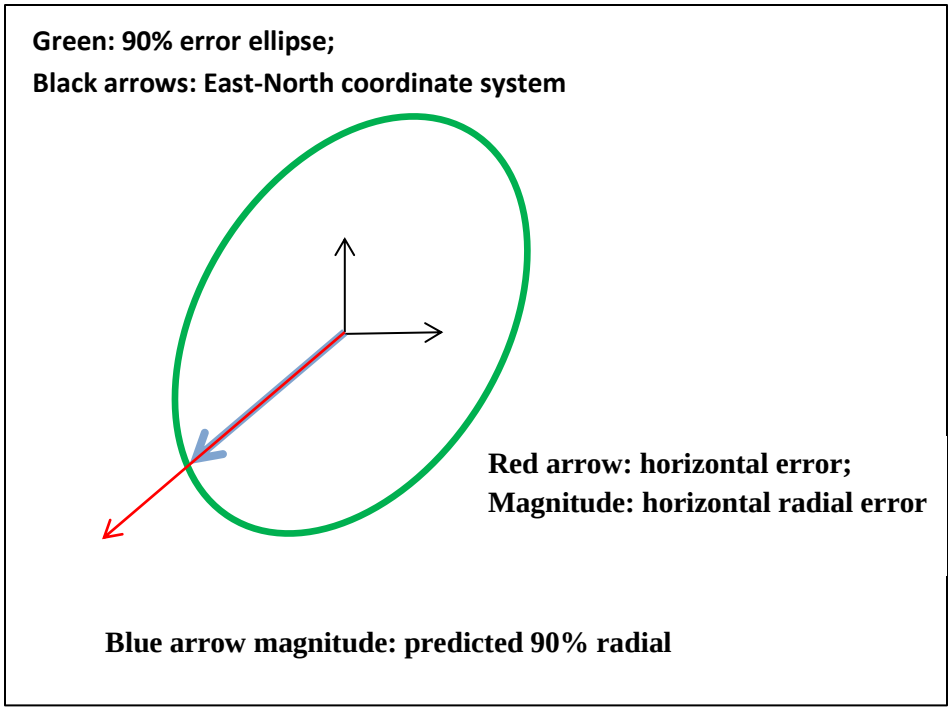


Figure 4.2-1: Example of horizontal radial error (excessively large in this particular example for clarity) versus corresponding predicted radial at the 90% probability level; the 90% predicted radials corresponding to independent samples of horizontal error will intersect the error ellipse at different locations along its boundary

An i.i.d. sample of horizontal radial error ϵh_{s_k} is computed from the horizontal error sample ϵX_{s_k} which is statistically consistent with the corresponding true (and unknown) error covariance matrix, whereas the corresponding computed predicted radial is consistent with the direction of ϵX_{s_k} and the corresponding predicted error covariance matrix. When the two error covariance matrices are reasonably similar and a reasonable number of i.i.d. samples are available, Equation (4.2-1) will be satisfied with a high level of confidence as illustrated later.

Note: An XX% (probability) error ellipse, together with its specified level of probability (XX%), is equivalent to the predicted error covariance matrix from which it is computed (see TGD 2a, Section 5.3 regarding an error ellipse and its corresponding error covariance matrix).

Where do the *normalized horizontal error tolerance requirements* come from?

The specified values $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$ of Equation (4.2-1) are "tailored" to both an assumed minimum number of i.i.d. error samples available to the corresponding validation process, and to a required "level of fidelity" of the underlying predicted statistical error model, i.e., the fidelity of predicted accuracy, or more specifically, the predicted error covariance matrix.

Level of fidelity is defined in terms of "sigma deviation" or sig_dev as follows:

Table 4.2-1: Predicted accuracy fidelity categories versus sigma deviation range

"sigma deviation" (%) per pred acc fidelity category		
high	medium	low
-5 to +5	-15 to +20	-30 to +40

Sigma deviation range is a convenient and approximate method to express the summed effects of various differences between a predicted error covariance matrix and the corresponding true (but unknown) error covariance matrix. Specifically, the $2x2$ predicted error covariance matrix (C_X_pred) is assumed to satisfy the following relationship with the $2x2$ true error covariance matrix (C_X_true):

$$(1 + sig\_dev_l)^2 C_X\_true \leq C_X\_pred \leq (1 + sig\_dev_r)^2 C_X\_true, \quad (4.2\text{-}3)$$

where sig_dev_l and sig_dev_r correspond to the left and right end points, respectively, of the sigma deviation range (interval) for the desired category of predicted accuracy fidelity per Table 4.2-1. The scalar multiplier (e.g. $(1 + sig\_dev_l)^2$) multiplies each component of C_X_true to yield a corresponding scaled true error covariance matrix.

The inequalities between covariance matrices in Equation (4.2-3) are important and useful relationships. In general, two covariance matrices satisfy $A \leq B$ if $(B - A) \geq 0$, i.e., the matrix $(B - A)$ is a positive semi-definite matrix. This is further described and detailed in Section 5.3.5 of TGD2a (predictive statistics), including the relationship between corresponding probability ellipses.

The above is illustrated graphically in Figure 4.2-2 below corresponding to medium predicted accuracy fidelity, where the probability ellipses represent and are equivalent to corresponding error covariance matrices. More specifically, medium predicted accuracy fidelity corresponds to $(1 - 0.15)^2 C_X\_true \leq C_X\_pred \leq (1 + 0.20)^2 C_X\_true$, per Equation (4.2-3) and Table 4.2-1. For convenience, this relationship is also stated as "predicted error covariance matrices are somewhere within $(1 - 0.15)^2$ to $(1 + 0.20)^2$ times the corresponding true error covariance matrices".

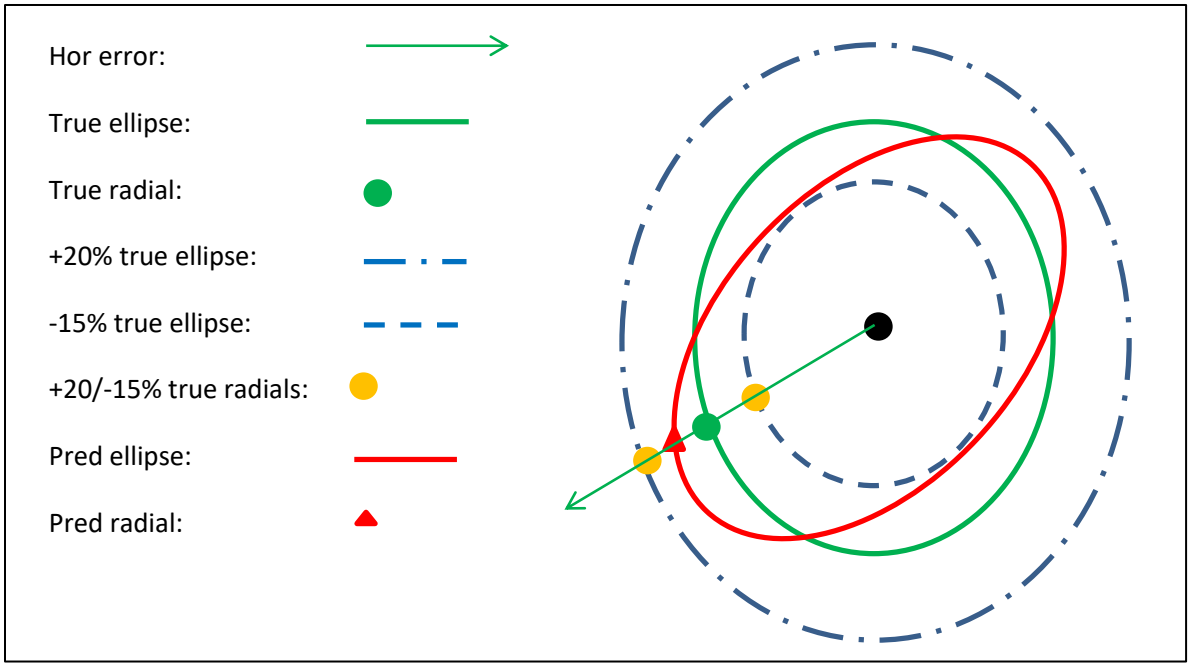


Figure 4.2-2: The predicted error covariance matrix C_X_pred (red ellipse) is bounded by scaled versions of the the true error covariance matrix C_X_true (blue ellipses) consistent with Equation (4.2.3)

In the above figure, the probability ellipses can be assumed to be at the 90% probability-level. However, as the figure presents a relative comparison, any probability-level is applicable as long as it is common to all of the ellipses. In general, a probability ellipse scales with probability-level.

The degree of maturity, level of system calibration, and the variability of the operational range of an NSG system affects the specified level of fidelity which is implicit in the specified values $\{YY_{h\_99\_spec}, YY_{h\_90\_spec}, YY_{h\_50\_spec}\}$ for flexibility. Recommended values are discussed in Section 5.4.2, with the following table representative:

Table 4.2-2: Normalized horizontal error tolerance values (%) vs. number of i.i.d. samples and predicted accuracy fidelity

normalized test level	400 samples	100 samples	50 samples	25 samples	pred acc fidelity
99	97	95	93	90	
90	85	83	78	76	high
50	44	39	38	34	
99	95	90	88	84	
90	78	76	72	68	medium
50	36	30	30	24	
99	85	84	82	76	
90	65	64	60	54	low
50	25	24	18	14	

The fewer the number of available samples and/or the lower the required predicted accuracy fidelity (NSG system specific), the lower (easier to pass) the normalized error test tolerances.

High predicted accuracy fidelity example

As an example, consider specifying predicted accuracy requirements for an NSG geolocation system based on same-pass stereo commercial satellite imagery. The system is intended for a high-value mission and utilizes data collected by a specific sensor platform with a proven CONOPs; therefore, high predicted accuracy fidelity is both desired and reasonable. The investment to obtain 100 i.i.d. error samples (minimum) for the validation of predicted accuracy is deemed reasonable as well. Hence, the values of the normalized error tolerances selected from Table 4.2-2 would nominally be $\{YY_{h\_99\_spec} = 95$, $YY_{h\_90\_spec} = 83$, $YY_{h\_50\_spec} = 39\}$, and their explcit values included in Equation (4.2-1) as part of the specification of predicted horizontal accuracy requirements.

High predicted accuracy fidelity corresponds to a sigma deviation range of -5% to +5% per Table 4.2-1. Correspondingly, and as detailed in Section 4.2.1 (Figure 4.2.1-1), if predicted error covariance matrices are somewhere within $(1 - 0.05)^2$ to $(1 + 0.05)^2$ times the corresponding true error covariance matrices, there is greater than a 90% probability that validation will pass as desired. On the other hand, if predicted error covariance matrices are less than approximately $(1 - 0.20)^2$ or greater than approximately $(1 + 0.25)^2$ times the corresponding true error covariance matrices, there is less than a 5% probability that validation will pass.

Low predicted accuracy fidelity example

As another and more extensive example that is at the "other end of the spectrum", consider specifying predicted accuracy requirements for an NSG geolocation system that is based on a tactical airborne sensor of specific sensor modality. Assume further that the underlying error models used in the extraction of a corresponding geolocation and its predicted error covariance matrix are relatively immature due to a combination of various factors, such as: (1) the complexity of errors affecting the geolocation results, (2) poor fidelity of sensor metadata predicted accuracies, (3) a wide-range for the constraints of sensor-to-geolocation geometries, and (4) a relatively non-mature CONOPS for the sensor/platform. Thus, low predicted accuracy fidelity is deemed applicable. Also, 100 samples are assumed available for subsequent validation processing. Hence, the values of the normalized error tolerances selected from Table 4.2-2 would nominally be $\{YY_{h\_99\_spec} = 84,\ YY_{h\_90\_spec} = 64,\ YY_{h\_50\_spec} = 24\}$.

Low predicted accuracy fidelity corresponds to a sigma deviation range of -30% to +40% per Table 4.2-1. Correspondingly, and as detailed in Section 4.2.1 (Figure 4.2.1-3), if predicted error covariance matrices are somewhere within $(1 - 0.30)^2$ to $(1 + 0.40)^2$ times the corresponding true error covariance matrices, there is greater than a 90% probability that validation will pass as desired. On the other hand, if predicted error covariance matrices are less than approximately $(1 - 0.40)^2$ or greater than approximately $(1 + 0.55)^2$ times the corresponding true error covariance matrices, there is less than a 5% probability that validation will pass.

Therefore, following successful validation of the NSG Geolocation System's predicted accuracy requirements, it is expected with reasonable confidence that a predicted error covariance matrix corresponding to an arbitrary geolocation (extraction) is between $(1-0.30)^2$ to $(1+0.40)^2$ times the true but unknown error covariance matrix. Correspondingly, a predicted $CE90$ computed from this predicted error covariance matrix is somewhere between 30% too small to 40% too large. Although this range of values is larger than desired, it is unavoidable and the NSG/user community's knowledge of its applicability is far more preferable than the various alternatives which include: (1) the non-computation of predicted accuracies, (2) the computation of predicted accuracies with no real pedigree, or (3) the specification of a higher predicted accuracy fidelity with the subsequent continued failure of its validation.

As a conservative option, if critical actions are based on the predicted scalar accuracy metric $CE90$, it can be subsequently scaled by a factor of approximately $1/(1-0.30)=1.43$. This is preferable over scaling the predicted error covariance itself, significantly reducing its information for other tasks, such as fusion. In addition, the (non-inflated) predicted error covariance still remains the best estimate of the true error covariance matrix.

Validation of predicted accuracy requirements is "plot friendly"

The overall specification and validation process for predicted accuracy is "plot-friendly", as illustrated in Figure 4.2-3 below which presents results corresponding to 100 i.i.d. error samples.

Each (non-normalized) horizontal radial error sample is plotted relative to the y-axis as a function of its corresponding predicted radial at the 90% probability level relative to the x-axis. The blue (middle) line corresponds to the 90% normalized error test, and the magenta (upper) and red (lower) lines to the 99% and 50% probability-level normalized error tests, respectively. Per Table 4.2-2, the percentage of blue circles under the magenta line must be greater than or equal to 95%, the percentage of blue circles under the blue line must be greater than or equal to 83%, and the percentage of blue circles above the red line must be greater than or equal to 39% in order to pass validation of predicted accuracy requirements. In this example, actual percentages were 97%, 87%, and 50%, respectively. Each of the three tests passed – validation of predicted horizontal accuracy was successful.

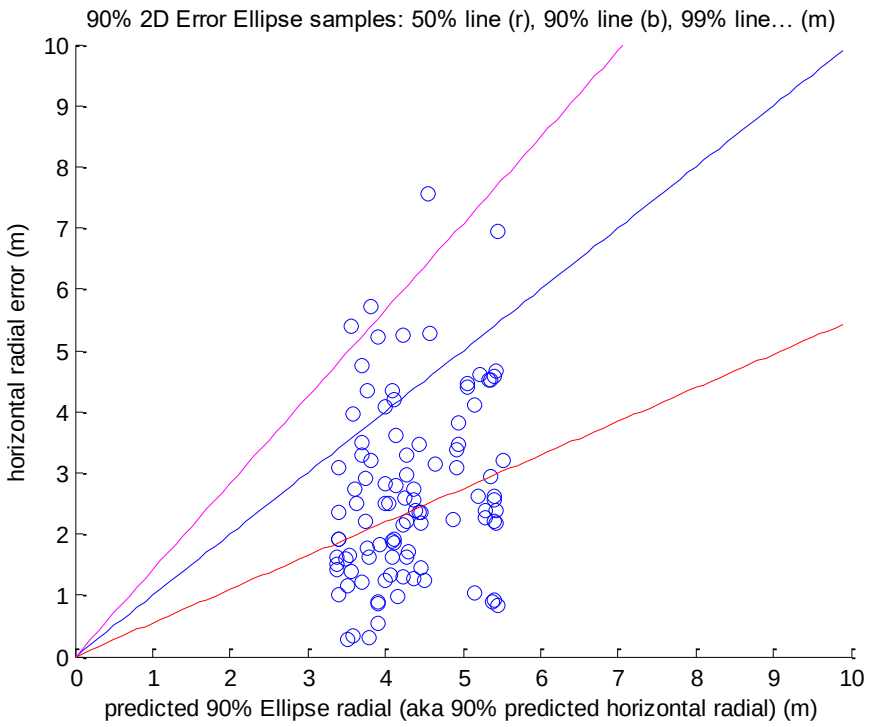


Figure 4.2-3: Graphical representation of the three normalized error tests corresponding to horizontal radial errors, 100 i.i.d. error samples, and high predicted accuracy fidelity

(Note: as illustrated in Figure 4.2-3, the blue line has a slope of 1 (or d_{90}/d_{90}) whereas the magenta and red lines have larger (d_{99}/d_{90}) and smaller (d_{50}/d_{90}) slopes, respectively. This is due to the plot's x-axis corresponding to 90% probability-level predicted radials for which the blue line is directly applicable – see Section 5.2.3 for additional details.)

The normalized error test at the 50% level corresponds to the percentage of (non-normalized) horizontal radial error samples above, not below, the red line. Thus, all three one-sided tests work "in concert" and in a practical manner to ensure that predicted horizontal accuracy values are neither too small nor too large relative to their corresponding radial errors. The "below the line" requirement for the normalized error tests at the 99% and 90% probability-levels, ensure that radial errors are not inconsistent with their predicted magnitudes (predicted radials) – not excessively large. The "above the line" requirement for the normalized error at the 50% level ensures that radial errors are not inconsistent with their predicted magnitudes – not excessively small. This is further illustrated conceptually in Figure 4.2-4 below. The 99% and 90% probability-level tests prevent the red-circle outliers (the 99% test the more extreme outliers), and the 50% probability-level test prevents the dark-red circle outliers. In particular, the latter test prevents artificial inflation of predicted error covariance matrices such that validation is virtually guaranteed to (incorrectly) pass.

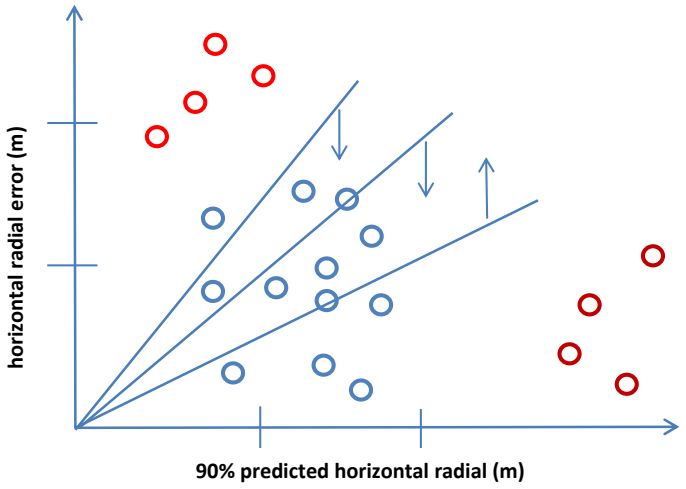


Figure 4.2-4: Conceptual graphic of the three probability-level normalized error tests for validation of predicted horizontal accuracy (all test lines blue)

Validation is based on predicted radials computed from the predicted error covariance matrix but scalar accuracy metrics can be substituted if necessary

The above specification and validation test of predicted accuracy utilized predicted radials to normalize radial error samples. These predicted radials were generated directly from the predicted (full) error covariance matrix – the baseline approach as detailed in Equation (4.2-2) and also termed "ellipsoidal-based" for reasons explained below. Predicted scalar accuracy metrics, such as $CE90$, can be used instead as the predicted radials, as documented later in Sections 5.2 and 5.4.3. However, even though predicted scalar accuracy metrics play an important overall role in the NSG (see TGD 1, Section 5.6), the baseline approach is preferred for the validation of predicted accuracy as the corresponding predicted radials are "closer" to the full-content of the predicted error covariance matrix [7].

This is demonstrated in Figure 4.2.5 below, which presents an example of a 0.9 probability error ellipse and a CE90 circle, both generated from the same error covariance matrix. As discussed in TGD 1, Section 5.5.1 , the error ellipse, along with its specified level of probability (0.9), is actually equivalent to the error covariance matrix – one can be uniquely derived from the other. However, as seen in the figure, the CE90 circle is not equivalent to the error ellipse – many different error ellipses correspond to the same CE90 circle, such as a rotated error ellipse, an error ellipse with smaller/larger semi-major/semi-minor axes of correct proportions (e.g., an error ellipse identical to the CE90 circle), etc. Therefore, the CE90 circle is not equivalent to the error covariance matrix.

Consequently, predicted radials generated from a predicted error covariance using the baseline approach will conform to the boundary of the error ellipse, as desired and as illustrated previously in Figure 4.2-1. However, CE90s, the radius of the CE90 circle, will not – their values are constant for a given predicted error covariance matrix: typically too big and occasionally too small per the ellipse boundary.

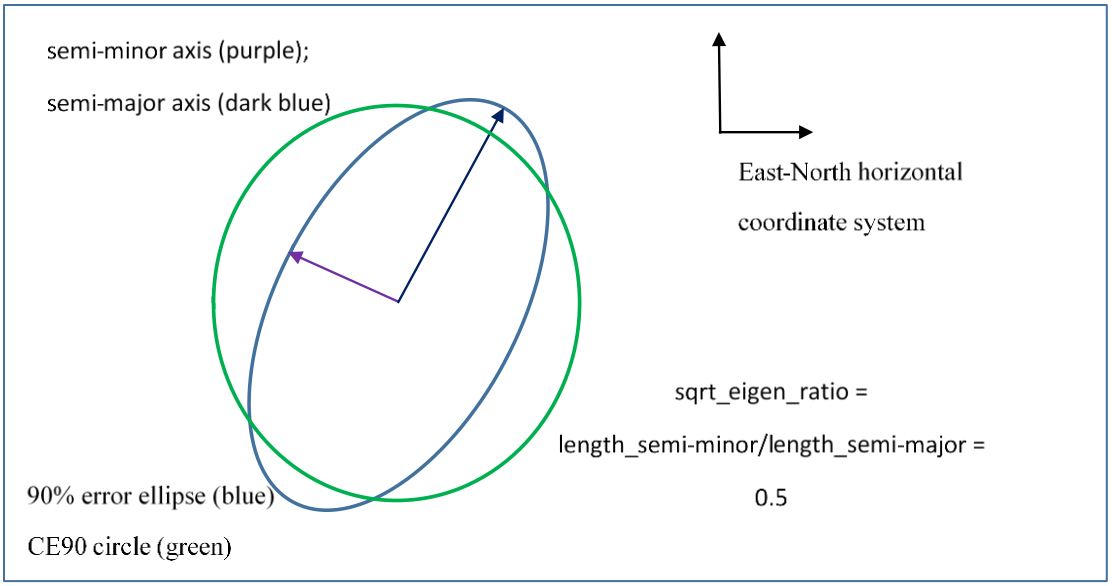


Figure 4.2-5: 90% error ellipse and corresponding CE90 circle; both generated using the same (full) error covariance matrix; both contain 0.90 probability, but the circle requires more area to do so

As discussed in Section 5.4.3, this is not a problem as long as the ellipse is not too elongated, i.e., if the ratio of its semi-minor to semi-major axis is approximately greater than 0.5. If this is not the case: (1) CE90 is not reasonably representative of actual predicted radials, and (2) the normalized error tolerance values $\{YY_{h\_99\_spec}\ , YY_{h\_90\_spec}\ , YY_{h\_50\_spec}\ \}$ corresponding to the use of CE90 are a function of this ratio and complicated to both derive and to utilize.

A specific example of the use of scalar accuracy metrics for the normalization of radial errors and corresponding plots is presented in Section 5.2.3.5 and corresponds to a ratio approximately equal to 0.8. It further notes that, unlike with the baseline method, a combined plot of the results of all three normalized error tests (magenta, blue, and red lines) is not applicable – three separate plots are required.

Finally, it must be emphasized that the non-baseline use of scalar accuracy metrics for the validation of predicted accuracy is to be implemented only if necessary: for example, if only scalar accuracy metrics computed from the full predicted error covariance matrix are available to the validation process – not the actual error covariance matrix itself.

4.2.1 Validation errors and confidence in passing validation versus sigma deviation

Finally, one last feature of the validation process for predicted accuracy is as follows: the three simultaneous normalized error tests at the different probability levels (99%, 90%, and 50%), along with the recommended values for their corresponding specified tolerances (see Table 4.2-2), are used in concert to minimize Type I validation errors and Type II validation errors. Type 1 validation errors correspond to predicted accuracy error covariance matrices consistent with the desired level of predicted accuracy fidelity, but at least one of the three normalized error validation tests fails; i.e., validation fails incorrectly. Type II validation errors correspond to predicted accuracy error covariance

matrices inconsistent with the desired level of predicted accuracy fidelity, yet all three normalized error tests pass; i.e., validation passes incorrectly.

This is illustrated in Figure 4.2.1-1 which presents the confidence in passing the overall predicted accuracy validation process versus "sigma deviation" of predicted error covariance matrices relative to the true (but unknown) actual error covariance matrices. For example, a sigma deviation of -30% corresponds to the predicted error covariance matrix equal to (1.0 - 0.30)^2 times (all elements or components of) the true error covariance matrix, i.e., optimistic predicted accuracy. A deviation of +10% corresponds to the predicted error covariance matrix equal to (1.0 + 0.10)^2 times the true error covariance matrix, i.e., pessimistic (conservative) predicted accuracy.

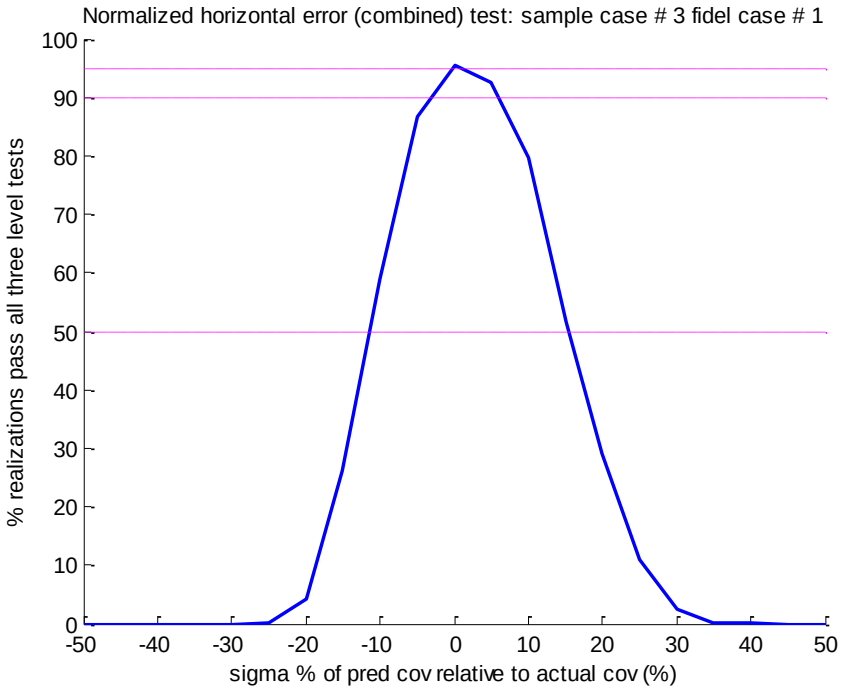


Figure 4.2.1-1: Confidence in passing all three normalized error tests vs. sigma deviation for horizontal errors: high predicted accuracy fidelity and the use of 100 i.i.d. error samples specified

In particular, Figure 4.2.1-1 corresponds to the use of 100 i.i.d. error samples and high predicted accuracy fidelity; thus, the values $\{YY_{h\_99\_spec} = 95\ , YY_{h\_90\_spec} = 83\ , YY_{h\_50\_spec} = 39\ \}$ were utilized per Table 4.2-2 for the normalized error tests as defined in Equation (4.2-1). Furthermore, high predicted accuracy fidelity is defined as corresponding to a sigma deviation equal to between -5% to +5% per Table 4.2-1, or more precisely, the predicted error covariance matrix satisfies the relationship: $(1 - sig\_dev_l)^2 C_X\_true \leq C_X\_pred \leq (1 + sig\_dev_r)^2 C_X\_true$. (For contrast, medium predicted accuracy fidelity is defined as corresponding to a sigma deviation between -15% to +20%.)

Note that in Figure 4.2.1-1 there is a confidence of at least 90% (second highest horizontal dotted magenta line) that validation will pass in the desired sigma deviation range (-5% to +5%) and very little

confidence (highly unlikely) that validation will pass outside of the desired sigma deviation range, as desired. Also, if a greater number of samples were used (and Table 4.2.2 entries selected appropriately), there would be less "roll-off" in the figure, i.e., confidence would drop-off faster after the range of desired fidelity is exceeded. This is illustrated in Figure 4.2.1-2, identical to Figure 4.2.1-1 except that 400 i.i.d. samples are applicable.

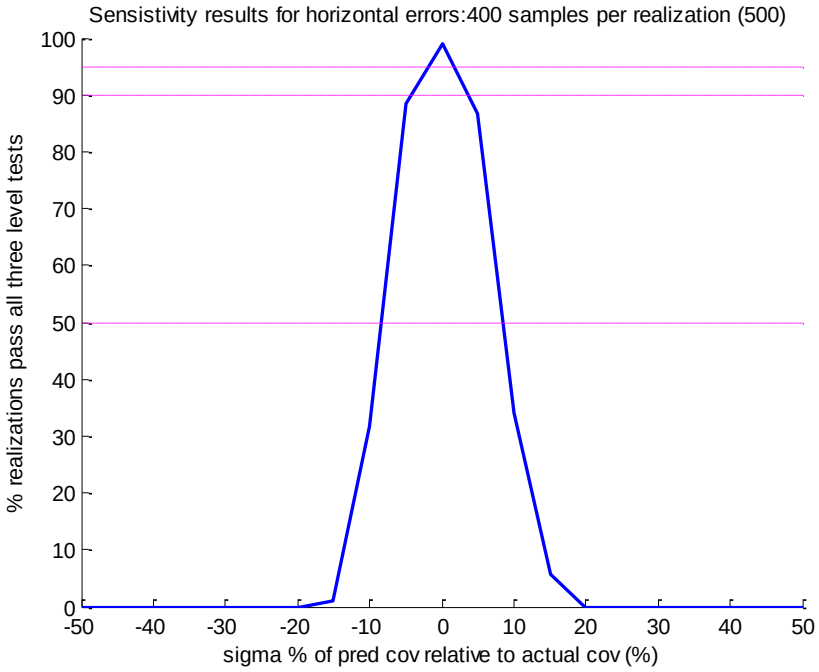


Figure 4.2.1-2: Confidence in passing all three normalized error tests vs. sigma deviation for horizontal errors: high predicted accuracy fidelity and the use of 400 i.i.d. error samples specified

Finally, if a lower level of predicted accuracy fidelity were specified (medium or low) as appropriate for a specific NSG-system, possibly due to significantly varied operational scenarios or error models (error propagation) with lower fidelity in the extraction process itself, the confidence would remain at least 90% over a larger range of sigma deviation consistent with the specified level of predicted accuracy fidelity. This is illustrated in Figure 4.2.1-3 for low predicted accuracy fidelity (sigma deviation from -30% to +40%) and 100 i.i.d. error samples.

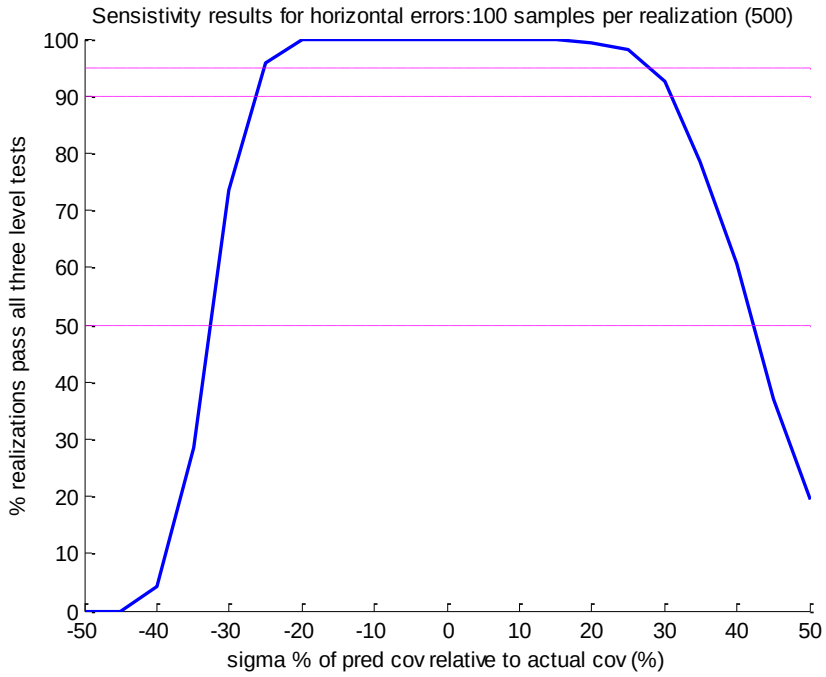


Figure 4.2.1-3: Confidence in passing all three normalized error tests vs. sigma deviation for horizontal errors: low predicted accuracy fidelity and the use of 100 i.i.d. error samples specified

Section 5.4.2 provides more details regarding the above plots and their correspondence to the level of predicted accuracy fidelity. It also discusses the levels of predicted accuracy fidelity per se, including the non-symmetric sigma deviation ranges corresponding to low and medium fidelity levels. Section 5.4.2 also contains more plots similar to the above plots but corresponding to different combinations of predicted accuracy fidelity and number of samples.

4.2.2 Recommended Number of Error Samples

At least 100 i.i.d. error samples are recommended for the validation of predicted horizontal accuracy regardless the desired level of predicted accuracy fidelity. This recommendation is consistent with the recommended number of i.i.d. error samples for accuracy (Section 4.1.1) and is also such that the aforementioned "roll-off" (Figure 4.2.1-1) is reasonably fast outside of the desired sigma deviation (predicted accuracy fidelity) range. (These same recommendations are applicable to the validation of predicted vertical and 3d accuracy requirements as well.)

As few as 40 i.i.d. errors samples, a firm minimum, can also be used for formal validation, with the corresponding restrictive caveat that "roll-off" or Type II validation errors will be larger than desired.

Type I and Type II predicted accuracy validation errors were discussed in the previous subsection and are more formally defined as follows:

Type I validation errors are defined as the event that actual predicted accuracy fidelity is as specified (low, medium, or high), but validation of predicted accuracy incorrectly fails.

Type II validation errors are defined as the event that actual predicted accuracy is not as specified (low, medium, or high), but validation of predicted accuracy incorrectly passes.

Minimization of Type I validation errors is a "given" by the method in which the normalized error tolerances were developed – there is a "built-in" probability or confidence of at least 90% that validation will pass when it should, and thus less than a 10% probability of a Type I validation error guaranteed. The values of the normalized error tolerances of Table 4.2-2 are consistent with this methodology - see Section 5.4 for further details.

Type II validation errors are minimized by the use of more i.i.d. samples – the larger the number of samples, the faster the "roll-off" in the confidence of passing validation after the desired sigma deviation range is exceeded as described in the previous subsection.

4.2.3 Pseudo-code for the Validation of Predicted Accuracy Requirements

Appendix C contains pseudo-code (MATLAB) that performs the entire predicted accuracy validation process given the appropriate inputs per accompanying documentation and in compliance with the above Section 4.2 overview and the corresponding details presented in Section 5.2. Pseudo-code output includes plots similar to Figure 4.2-3 presented earlier. Appendix C also contains examples of the pseudo-code's explicit use.

4.3 Guide to Detailed Technical Content

The following presents a guide to Section 5, references provided in Section 7, and the various appendices of this document. After this more "formal" guide, a "practical" guide is also presented that identifies recommended (sub)sections to the general reader (non-implementer) and those that can be by-passed, if so desired.

When a section is referenced, all of its underlying (sub)sections are assumed to be referenced as well, unless specifically stated otherwise.

- **Section 5.1** describes how to **specify and validate** (absolute) **accuracy** requirements, and includes various Monte-Carlo simulation-based examples. Portions of this section are in "specification-like" and "bulletized" form in order to present details in a concise manner and appropriate format. Section 5.1 is the detailed companion to Section 4.1 already presented.
- **Section 5.2** describes how to **specify and validate** (absolute) **predicted accuracy** requirements, and includes various Monte-Carlo simulation-based examples. Portions of this section are in "specification-like" and "bulletized" form in order to present details in a concise manner and appropriate format. Section 5.2 is the detailed companion to Section 4.2 already presented.
- **Section 5.3** describes how to **specify and validate relative accuracy** and **predicted relative accuracy** requirements. As opposed to the explicit details of Sections 5.1 and 5.2, for document efficiency it summarizes changes required to these earlier sections such that they are applicable to relative accuracy.
- **Section 5.4** discusses and presents recommended and explicit values for the **normalized error test tolerances** at the 99%, 90% and 50% probability-levels, which are key to the specification and validation of predicted accuracy requirements and referenced in both Sections 5.2 and 5.3.

These values take into account both the expected number of i.i.d. samples available to validation, and the desired (specified) degree of fidelity of the predicted statistical error models in the corresponding NSG system. The latter is directly related to the fidelity of predicted accuracy and is categorized by simple and convenient "high", "medium", and "low" summary descriptors. This section also discusses recommended future research.

- **Section 5.5** discusses the relationships between the various predicted accuracy validation tests, the plotting of error samples versus their corresponding predicted accuracy, and various probability distributions of related random variables, including the Chi-square distribution.
- **Section 5.6** discusses the appropriate use (when/how) of extra but **correlated error samples** in validation.
- **Section 5.7** discusses the desired characteristics of **ground truth** and their effect on validation.
- **References (Section 7)** include those directly related to the specification and validation of accuracy [7], ground truth [1], [4], [6], and design of experiment for the evaluation of tracking system accuracy and corresponding characterization [3]. In addition, Reference [5] presents an alternate overview of the contents of this document.
    - Reference [3] is rather extensive, includes theoretical derivations, and concentrates on the characterization of accuracy using parametric model-building and the computation of sample-based statistics of Target Location Errors (TLE) as opposed to specification and validation of accuracy per se. Sample-based statistics include scalar accuracy metrics at multiple levels of probability and (direction dependent) error ellipsoids previously discussed in TGD 1, TGD 2a, and TGD 2b.
    - Reference [5] summarizes Sections 4, 4.1, and 4.2 of this document in a somewhat easier to read fashion with less detail and a slightly different persepective.

  The references listed in Section 7 are in addition to other TGD documents which are referenced throughout Section 4 and Section 5.
- **Appendices** consist primarily of pseudo-code (MATLAB) that support both the various investigations and examples contained in Section 4 and Section 5, as well as explicit pseudo-code for "formal" validation of both accuracy and predicted accuracy requirements.

Practical guide

The following assumes that Section 4 has already been read. Furthermore, some of the material is unique to Section 4, such as the recommended number of i.i.d. error samples discussed in Sections 4.1 and 4.2:

Much of the material in Sections 5.1 and 5.2 has already been covered in Sections 4.1 and 4.2, respectively, but is now covered at a more detailed level, including more examples. As such, Sections 5.1 and 5.2 may not be of interest to the general reader (non-implementer), other than the material at the start of Section 5 itself which is recommended to all. In particular, it discusses the operational scenario for geolocation extraction; its top-level description should be included as part of a specification for accuracy and/or predicted accuracy.

Section 5.3 discusses relative accuracy and predicted relative accuracy and their validation and may not be of interest to the general reader – it is also somewhat succinct as it refers to concepts/terminology detailed in Sections 5.1 and 5.2, now expanded in order to address relative accuracy.

Section 5.4 presents important additional information regarding the definition of the various classes of predicted accuracy fidelity (low, medium, high) and normalized error tolerance values and their effect on the specification and validation of predicted accuracy. In addition, Section 5.4.3 discusses the non-baseline use of scalar accuracy metrics (e.g., $CEXX$) instead of the error ellipsoid (predicted full error covariance matrix) for the generation of predicted radials in the validation of predicted accuracy. Section 5.4.4 discusses recommended future research associated with the classification of predicted accuracy fidelity, the "tuning" of baseline values for normalized error tolerances, and the assumption of a multi-variate Gaussian distribution of underlying errors. Section 5.4 is recommended to all, other than Section 5.4.3 which may only be of interest if the non-basline approach is to be implemented.

Section 5.5 presents more background theory regarding the approach to the specification and validation of predicted accuracy. This subsection is of interest to those concerned with underlying theory, and may not be of interest to the general reader.

Sections 5.6 and 5.7 discuss important concepts and procedures not addressed in either Section 4 or elsewhere in Section 5 regarding the possible use of correlated error samples during validation (Section 5.6), and the use/effect of ground truth points in validation (Section 5.7), respectively. Subsection 5.6 may not be of interest to the general reader, other than its beginning which is recommended to all. Section 5.7 is recommended to all.

In summary, the following sections are recommended to all:

- Section 4 (assumed already read)
- Section 5 (beginning only), Section 5.4 (Section 5.4.3 optional), Section 5.6 (beginning only), Section 5.7

Sections that were not listed directly above are also recommended to implementers/developers of specifications, and in particular, to implementers of validation procedures. They may be by-passed by the general reader, if so desired. Various appendices are recommended to those interested per their references throughout the main body of the document.

5 Methodologies and Algorithms for Specification and Validation of Accuracy and Predicted Accuracy in the NSG

This section presents detailed methodologies, definitions, equations and algorithms for both the specification and validation of accuracy requirements and predicted accuracy requirements.

The following generic forms for the specification and validation of accuracy requirements (Section 5.1) and the specification and validation of predicted accuracy requirements (Section 5.2) are applicable to the extraction of geolocations. A relevant geolocation is typically a ground location (coordinates) of a feature of interest. All that is specifically assumed is that extractions are (near) optimal estimates of the geolocations, such as outputs from a properly modelled Weighted Least Squares estimator, using corresponding measurements in applicable sensor data, and which includes the corresponding solution (*a posteriori*) error covariance matrices. As typical of most estimators, the estimator is assumed unbiased, i.e., solution errors have a mean-value of zero.

A description of the appropriate operational scenario or constraints associated with the relevant extractions is also included in or pointed to by both the specification of accuracy and the specification of predicted accuracy requirements, including the assumed number of sensor measurements used by the extraction, their inter-related collection constraints, and their assumed explicit measurement error (one-sigma). These are NSG geolocation system-specific and will vary by system. They are not specified in the following descriptions (Sections 5.1 and 5.2) per se. If for a given NSG geolocation system, operational constraints (typically sensor-to-geolocation of interest geometry or angles) are highly varied with corresponding accuracy varied in accordance, the operational constraints can be categorized with a separate set of accuracy requirements and validation applicable to each category (e.g. range of angles). However, there is an associated NSG trade-off: many categories allow for “tighter” accuracy requirements per category, but can lead to too few i.i.d. error samples available per category for validation.

Examples of two different operational scenarios of NSG geolocation systems based on commercial EO satellite imagery provide NSG-relevant illustrations as follows: (1) MIG extractions of geolocation based on two (stereo) same-pass images, and (2) MIG extractions of a geolocation based on one (mono) image with an external elevation source. MIG extractions are Multi-Image Geolocation (MIG) extractions that are based on Weighted Least Squares, include rigorous error propagation, and in particular, include the (*a posteriori*) error covariance matrix of the geolocation solution, i.e., the predicted error covariance matrix. Section 5.8.1.1 of TGD 1 describes MIG in more detail, including the possible use of only one image, i.e., “Multi-Image” in Multi-Image Geolocation extraction is a general term.

Both of the above operational scenarios assume explicit measurement (mensuration) errors of one pixel (one-sigma) for both the line and sample image coordinates corresponding to the feature of interest identified in each image. In addition, the description of each scenario’s specific collection geometry constraints, such as the allowed range of off-nadir imaging angles, etc., is provided in the specifications of accuracy and predicted accuracy requirements.

An example of a possible set of collection geometry constraints for commercial satellite EO stereo imagery is as follows [8]: (1) convergence angle between 30 and 45 degrees, and (2) (off-nadir) roll angle between 0 and 20 degrees, and (3) asymmetry angle between 0 and 10 degrees. One set of specific accuracy requirements and one set of specific predicted accuracy requirements would be associated with this set of collection geometry constraints. Furthermore, only one set of constraints and corresponding accuracy and predicted accuracy specifications would be needed, as collection geometry is tightly controlled by the image provider.

It is assumed that for a particular NSG geolocation system, both (absolute) accuracy and predicted (absolute) accuracy are specified and are to be validated. As an option, relative accuracy and predicted relative accuracy (Section 5.3) can also be specified and validated. For NSG geolocation systems based on commercial EO satellite imagery, relative accuracy requirements usually correspond to monoscopic (one-image plus elevation source) MIG extractions.

Although the above examples correspond to image-based extraction of geolocation for ease of illustration, all related principles are equally applicable to the extraction of geolocations in general, regardless the type of sensor platform and/or sensor data. As stated above, all that is specifically assumed is that extractions are based on (near) optimal estimates (e.g., Weighted Least Squares estimates) of geolocation using applicable sensor measurements, and include corresponding solution (*a posteriori*) error covariance matrices. For convenience, we refer to these type of extractions as **"MIG-type" extractions** in remaining subsections regardless the type of sensor platform and/or sensor data.

5.1 Specification and Validation of Accuracy Requirements

This section describes the recommended form and content for the specification of accuracy requirements and their corresponding validation. The accuracy specification corresponds to any combination of vertical, horizontal, and 3d accuracy requirements. (Note: 3d accuracy is sometimes referred to as spherical accuracy.)

5.1.1 Accuracy Specification

General statement: NSG geolocation system xxxx shall satisfy the following accuracy requirements:

Explicit form:

- $\epsilon v_{XX} \leq LEXX_{spec}$ and/or (5.1.1-1)
- $\epsilon h_{XX} \leq CEXX_{spec}$ and/or
- $\epsilon r_{XX} \leq SEXX_{spec}$

Values specified:

- Probability level XX in %: 50, 90, or 95 (specify one, typically 90%).
- Scalar accuracy metric requirements at the XX% probability level: Linear Error $LEXX_{spec}$ and/or Circular Error $CEXX_{spec}$ and/or Spherical Error $SEXX_{spec}$ – values specified in meters (specify one, two, or three per "and/or" of Equation (5.1.1-1)).
- Description of operational constraints for applicable NSG geolocation system.

Underlying Definitions and Computations:

- Vertical radial error $\epsilon v = \sqrt{\epsilon z^2}$, horizontal radial error $\epsilon h = \sqrt{\epsilon x^2 + \epsilon y^2}$, and 3d radial error $\epsilon r = \sqrt{\epsilon x^2 + \epsilon y^2 + \epsilon z^2}$ in meters.
- $\epsilon x, \epsilon y, \epsilon z$ correspond to errors in x, y, z geolocation coordinates, respectively, as represented in an appropriate ENU coordinate system.
- ϵv_{XX} is the XX percentile of the random variable ϵv, i.e., $prob\{\epsilon v \leq \epsilon v_{XX}\} = 0.XX$;
  ϵh_{XX} is the XX percentile of the random variable ϵh;
  ϵr_{XX} is the XX percentile of the random variable ϵr;
  all XX percentiles have values in meters.
  - These percentiles are also termed XX % vertical Linear Error ($LEXX$), horizontal Circular Error ($CEXX$) and 3d or Spherical Error ($SEXX$), respectively.
- $prob$ (probability) is taken over all possible MIG-type extractions within operational constraints for the corresponding NSG geolocation system.

Recommended but optional additions:

- Minimum number of i.i.d. error samples for corresponding formal validation of the accuracy requirement (specification) – value specified as $n\_iid\_min\_acc\_val$ (unitless).
- Confidence level for the corresponding formal validation of the accuracy requirement – value specified as $conf\_acc\_val$ in %: 50, 90, or 95 (specify one, typically 90%).

Other optional additions:

- Error bound (max none to exceed): $LEmax_{spec}$ and/or $CEmax_{spec}$ and/or $SEmax_{spec}$ – values specified in meters (specify one, two, or three values per "and/or" of Equation (5.1.1-1)). Corresponding definition assuming horizontal radial errors, and directly analogous for vertical and 3d radial errors:
  - $\epsilon h \leq CE_{spec\_max}$, (5.1.1-2)
    required for every MIG-type extraction within operational constraints;
  - Used to prevent very large error outliers in the NSG geolocation system (affects Quality Assurance requirements). Also, $CE_{spec\_max} \gg CEXX_{spec}$ for specified probability XX. Similar definitions and relationships are applicable for $LEmax_{spec}$ and $SEmax_{spec}$.
- Multiple sets of the accuracy specification, one for each corresponding and identified set of operational constraints; of possible use for geolocation systems with highly varied operational constraints.

Comments:

- The explicit form for the accuracy specification in Equation (5.1.1-1) allows for any combination (one, two, or three) of vertical, horizontal, and 3d accuracy requirements.

- The accuracy requirements of Equation (5.1.1-1) can be written in the equivalent alternate form:
  - $prob\{\epsilon v \leq LEXX_{spec}\} \geq 0.XX$ and/or (5.1.1-3)
  - $prob\{\epsilon h \leq CEXX_{spec}\} \geq 0.XX$ and/or
  - $prob\{\epsilon r \leq SEXX_{spec}\} \geq 0.XX$.
- The above alternate form for the accuracy specification can also be considered equivalent to the following "less succinct" form, as illustrated for horizontal errors:
  - $CEXX \leq CEXX_{spec}$, where $CEXX$ is considered taken over all arbitrary MIG-type extractions within the operational constraints, i.e., "$prob$" is "built-in" to $CEXX$, and $CEXX \equiv \epsilon h_{XX}$.
- The appropriate values for the core accuracy requirements (e.g., $CEXX_{spec}$) typically correspond to a system-level error analysis and flow-down of requirements that are performed as part of system conceptual design, and further refined during system design and then initial operations.
- The specified requirements $LEXX_{spec}$, $CEXX_{spec}$, and/or $SEXX_{spec}$ may correspond directly or indirectly to specified targeting accuracy requirements, CAT-level I, II,…, VI, corresponding to Target Location Error (TLE), if so desired.
- It is possible that the values of $LEXX_{spec}$, $CEXX_{spec}$, and/or $SEXX_{spec}$ are to be converted from available $LEYY_{spec}$, $CEYY_{spec}$, and/or $SEYY_{spec}$ for convenience, where $XX \neq YY$. Conversion factors are presented in Section I.2 of Appendix I.

5.1.2 Validation of Specified Accuracy Requirements

General statement: The accuracy requirements for NSG geolocation system xxxx shall be validated as follows:

Explicit form:

- $\epsilon v_{XX} \leq LEXX_{spec}$ validated if: and/or (5.1.2-1)
  - $lub\_\epsilon v_{XX} \leq LEXX_{spec}$
- $\epsilon h_{XX} \leq CEXX_{spec}$ validated if: and/or
  - $lub\_\epsilon h_{XX} \leq CEXX_{spec}$
- $\epsilon r_{XX} \leq SEXX_{spec}$ validated if:
  - $lub\_\epsilon r_{XX} \leq SEXX_{spec}$

Values specified:

- Probability level XX in %: 50, 90, or 95 (specify one per Equation (5.1.2-1) consistent with Accuracy Specification's Equation (5.1.1-1)).
- Scalar accuracy metric requirements at the XX% probability level: $LEXX_{spec}$ and/or $CEXX_{spec}$ and/or $SEXX_{spec}$ – values specified in meters (specify one, two, or three per Equation (5.1.2-1) and consistent with the Accuracy Specification's Equation (5.1.1-1)).
- Optional: Minimum number of i.i.d. error samples for validation of the Accuracy Specification – value specified as $n\_iid\_min\_acc\_val$ (unit-less) per Accuracy Specification of Section 5.1.1.
- Optional: Confidence level for the corresponding validation of the Accuracy Specification – value specified as $conf\_acc\_val$ in %: 50, 90, or 95 (specify one) per Accuracy Specification of Section 5.1.1.

Underlying Definitions and Computations:

- The tests of Equation (5.1.2-1) are based on the computation of a percentile's "least-upper-bound" in meters (e.g., $lub\_\epsilon h_{XX}$ (meters)) based on order statistics of i.i.d. error samples at a (minimum) specified level of confidence of $YY\%$ (aka confidence-level or coefficient γ), as defined in TGD 2b on sample statistics:
    - For example, $lub\_\epsilon h_{XX}$ is defined as the smallest value such that the (true and unknown) percentile ϵh_{XX} is less than $lub\_\epsilon h_{XX}$ with probability of at least $YY\%$, i.e., $prob\{\epsilon h_{XX} \leq lub\_\epsilon h_{XX}\} \geq 0.YY$.
    - The actual value of $lub\_\epsilon h_{XX}$ computed via order statistics is the value of the smallest ordered sample such that it is greater than or equal to ϵh_{XX} with at least probability $YY\%$.
- YY is equal to 50, 90, or 95%; its value may have been specified as part of the corresponding Accuracy Specification, in which case $YY = conf\_acc\_val$, otherwise $YY = 90$ % is the default confidence level. Note that the value of the confidence YY % need not be the same as the value of the percentile's probability $XX\%$.
- Enough i.i.d. samples must be used such that a least-upper-bound is valid, i.e., its computed value is not contained in the open interval greater than the largest order sample value (see Section 5.3 of TGD 2b). In addition, the tests of Equation (5.1.2-1) can only be formally performed if the number of i.i.d. samples is at least equal to $n\_iid\_min\_acc\_val$, the optional value from the corresponding Accuracy Specification.
- Either a one-sided confidence interval or the right end-point of a two sided confidence interval can be used as the least-upper-bound at a specified (minimum) level of confidence. The two different confidence intervals are almost always equal if the number of i.i.d. samples is greater than 50, although when different, the one-sided is preferred. Also, the one-sided confidence interval corresponds to the actual definition of (probabilistic) least-upper-bound regardless the number of samples. See Section 5.3 of TGD 2b for a complete description of applicable order statistics and Sections 5.3.3.1, 5.3.6, and 5.4.1 of TGD 2b in particular regarding the equivalence of a one-sided confidence interval with the least-upper-bound and the use of a two-sided confidence interval as an approximation. The pseudo-code for the validation of accuracy contained in Appendix B incorporates one-sided confidence intervals as the default.
- The application of order statistics (see Section 5.3 of TGD 2b) is applicable to a scalar random variable, and for the application of interest in this document, either vertical, horizontal, or 3d radial errors – the specific type of radial error is immaterial. For example, given 100 i.i.d. vertical radial error samples, the best estimate of the 90th percentile of vertical radial error is equal to the value of the 90th ordered sample. Given 100 i.i.d. horizontal radial error samples instead, the best estimate of the 90th percentile of horizontal radial error is also equal to the value of the 90th ordered sample.

Comments:

- In general, the more i.i.d. samples used in the underlying computations for the tests of Equation (5.1.2-1), the better the Validation.

- The use of order statistics requires no assumption regarding the probability distribution of underlying errors, including whether their distribution is Gaussian or not or whether its mean-value is zero or not – a very desirable feature, particularly for a general specification.
- If only a very small number of i.i.d. samples are available (e.g. 20) and validation must be performed, at least an initial validation, order statistics are not applicable. However, Section 5.4.2 of TGD2b presents an alternative approach using classical sample statistics, i.e., the sample mean, sample standard deviation, and confidence intervals.
- In addition to the above validation processing (Equation (5.1.2-1)), verification of a non-bias in each relevant error component (e.g., ϵx,ϵy,ϵz) is recommended based on various techniques presented in Section 5.2 of TGD2b (sample mean, hypothesis test, etc.). If a significant bias is found, it should be reported, regardless the formal validation results, for corrective action. These techniques can also be augmented in order to better characterize errors and their sources by their representation in different coordinate systems that are sensor-dependent, such as a Cartesian coordinate system centered at the extracted point with the coordinate system's z-axis aligned with the sensor's line-of-sight direction [3].

5.1.3 Example of the Specification of Accuracy Requirements and Corresponding Validation

All of the specific numbers in the following example are for illustrative purposes only, and roughly approximate those applicable to geolocations solved for by a MIG-type extraction using stereo images from a commercial EO imaging satellite for specificity.

5.1.3.1 Accuracy Specification Example

- $\epsilon v_{90} \leq LE90_{spec} = 5.5\ meters$ and (5.1.3.1-1)
- $\epsilon h_{90} \leq CE90_{spec} = 5.5\ meters$

- Selected option: Error bound (max none to exceed) –
  - $LEmax_{spec} = 15\ meters$ (5.1.3.1-2)
  - $CEmax_{spec} = 15\ meters$

- Operational constraints: Above requirements are applicable to MIG-type extractions of two (stereo) same-pass images with corresponding imaging geometry: max off-nadir angle=…, minimum convergence angle=…, minimum bisector elevation angle (BIE)=…, maximum asymmetry angle=…, etc. Extraction mensuration errors are assumed equal to 1 pixel (one-sigma) in each of the image line and sample directions.
- Validation of the above is to be performed at a $conf\_acc\_val$ = 90% confidence level and with a minimum number of i.i.d. error samples $n\_iid\_min\_acc\_val$ = 95.

5.1.3.2 Corresponding Validation Example

Error samples $\epsilon X_{s\_k} = [\epsilon x_{s\_k} \quad \epsilon y_{s\_k} \quad \epsilon z_{s\_k}]^T$, $k = 1,..,100$, were first simulated corresponding to 100 i.i.d. samples of 3d geolocation error relative to ground truth (as opposed to using actual or "field" error samples). Validation processing then generated 100 corresponding vertical radial error samples: $\epsilon v_{s\_k} = \sqrt{\epsilon z_{s\_k}^2}$, $k = 1,..100$. Validation processing then also generated 100 corresponding horizontal radial error samples: $\epsilon h_{s\_k} = \sqrt{\epsilon x_{s\_k}^2 + \epsilon y_{s\_k}^2}$, $k = 1,..,100$. It then re-ordered both sets of radial error samples by ascending magnitude.

From the TGD 2b Section 5.3 on order statistics, we know that for 100 i.i.d. samples and for a 90% confidence level for the least-upper-bound, the least-upper-bound for the 90th percentile of vertical radial error is equal to the 95th ordered samples ϵv_{s_95}, and the least-upper bound-for the 90th percentile of horizontal radial error is equal to the 95th ordered samples ϵh_{s_95}. That is, $lub\_\epsilon v_{90} = \epsilon v_{s\_95}$ and $lub\_\epsilon h_{90} = \epsilon h_{s\_95}$. (This particular least-upper-bound corresponds to both a one-sided confidence interval as well as the right end-point of a two-sided confidence interval as they are equal for 100 samples and a 90% confidence level and a 90% percentile.)

Results of the simulation, including evaluation of Equation (5.1.2-1) are as follows:

$lub\_\epsilon v_{90} = 4.16\ meters < LE90_{spec} = 5.5\ meters$, therefore, the validation test passed;

$lub\_\epsilon h_{90} = 4.85\ meters < CE90_{spec} = 5.5\ meters$, therefore, the validation test passed.

And since both validation tests passed, overall validation of the accuracy requirements was successful. The non-ordered vertical error samples ϵv_{s_k} and the non-ordered horizontal errors samples ϵh_{s_k} are also plotted versus sample number in Figures 5.1.3.2-1 and 5.1.3.2-2, respectively. The horizontal solid magenta (upper) line corresponds to the least-upper-bound of the 90th percentile at a 90% confidence level, and the horizontal dotted red (lower) line corresponds to the best estimate of the 90th percentile for added information. (Note: the plots also clearly indicate that the error bound requirements of Equation (5.1.1-2) were met as well.) See Appendix F for the pseudo-code (MATLAB) that was used to perform the above simulation: generate error samples, evaluate Equation (5.1.2-1) and plot the results.

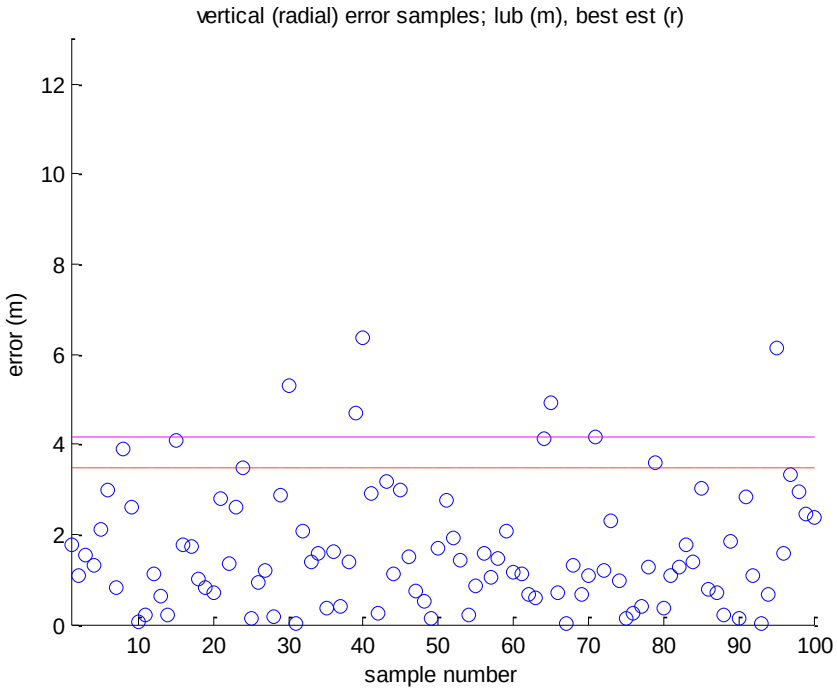


Figure 5.1.3.2-1: Vertical Radial Error Samples: magenta line is $lub\_\epsilon v_{90}$ at a 90% confidence level, red dotted line is best estimate of ϵv_{90} for added information; based on 100 i.i.d. error samples; compare $lub\_\epsilon v_{90}$ to corresponding specified requirement $LE90_{spec} = 5.5\ meters$

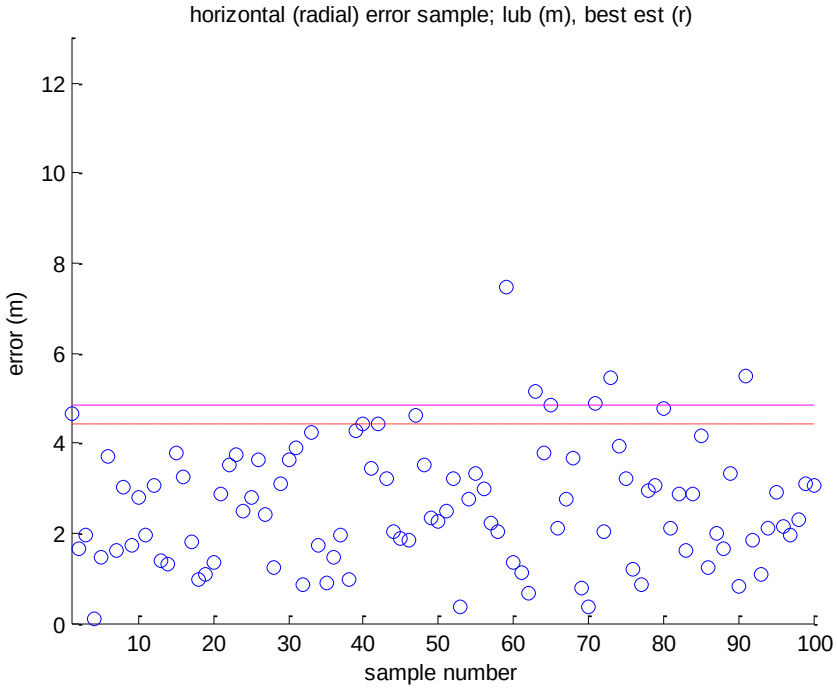


Figure 5.1.3.2-2: Horizontal Radial Error Samples: magenta line is $lub\_\epsilon h_{90}$ at a 90% confidence level, red dotted line is best estimate of ϵh_{90} for added information; based on 100 i.i.d. error samples; compare $lub\_\epsilon h_{90}$ to corresponding specified requirement $CE90_{spec} = 5.5\ meters$

Because the above example was based on simulation, we know that the true value for the vertical 90th percentile ϵv_{90} (or $LE90$) was approximately 3.5 meters, and the true value for the horizontal 90th percentile ϵh_{90} (or $CE90$) was approximately 4.6 meters. Thus, the result of "passing" validation was indeed correct. Further note that the true values of the percentiles were less than the required (specified) values of the percentiles, which should also be the case for NSG geolocation systems in general, i.e., a bit of "pad" or margin should be deliberately built into a reasonable specification for accuracy, particularly to account for only a reasonable number of i.i.d. error samples for corresponding validation.

The above tests were based on least-upper-bounds to correctly and automatically account for the effects of statistical significance due to a finite number of samples. The best estimate of the percentiles (as opposed to their least-upper-bounds) were also computed for additional information using order statistics as detailed in Section 5.3 of TGD 2b. For this example, they are equal to the 90th ordered samples, i.e., the best estimate of ϵv_{90} is equal to $\epsilon v_{s\_90} = 3.49$ meters and the best estimate of ϵh_{90} is equal to $\epsilon h_{s\_90} = 4.42$ meter, as indicated in Figures 5.1.3.2-1 and 5.1.3.2-2.

For a given confidence level, the greater the number of i.i.d. samples the "better" the validation via the least-upper-bound. For example, for a 90% confidence level and 100 samples, $lub\_\epsilon h_{90}$ corresponds to the 95th ordered sample, or 95% of the total number of ordered samples. For 200 samples, $lub\_\epsilon h_{90}$ corresponds to the 186th ordered sample, or 93% of the total number of ordered samples. There is less (relative) "pad" in the least-upper-bound when using 200 samples as opposed to using 100 samples in order to account for a finite number of samples.

For the formal validation of accuracy requirements for a commercial satellite EO imaging system, we expect the number of samples to be closer to 200 than to 100, the latter used in this simulation-based example for convenience.

5.2 Specification and Validation of Predicted Accuracy Requirements

This section describes the recommended form and content for the specification of predicted accuracy requirements and their corresponding validation. It is necessarily somewhat more complex than for accuracy requirements as it involves both error samples and their corresponding sample-specific predicted accuracy (error covariance matrix) from the MIG-type extractions.

The predicted accuracy specification corresponds to any combination of predicted vertical, horizontal, and 3d accuracy.

5.2.1 Predicted Accuracy Specification

General statement: NSG geolocation system xxxx shall satisfy the following predicted accuracy requirements:

Explicit form:

- Normalized vertical error and/or (5.2.1-1)
  - $prob\{\epsilon v\_norm_{99} \leq 1\} \geq 0.YY_{v\_99\_spec}$
  - $prob\{\epsilon v\_norm_{90} \leq 1\} \geq 0.YY_{v\_90\_spec}$
  - $prob\{\epsilon v\_norm_{50} > 1\} \geq 0.YY_{v\_50\_spec}$
- Normalized horizontal error and/or
  - $prob\{\epsilon h\_norm_{99} \leq 1\} \geq 0.YY_{h\_99\_spec}$
  - $prob\{\epsilon h\_norm_{90} \leq 1\} \geq 0.YY_{h\_90\_spec}$
  - $prob\{\epsilon h\_norm_{50} > 1\} \geq 0.YY_{h\_50\_spec}$
- Normalized 3d error
  - $prob\{\epsilon r\_norm_{99} \leq 1\} \geq 0.YY_{r\_99\_spec}$
  - $prob\{\epsilon r\_norm_{90} \leq 1\} \geq 0.YY_{r\_90\_spec}$
  - $prob\{\epsilon r\_norm_{50} > 1\} \geq 0.YY_{r\_50\_spec}$

Values specified:

- Normalized error tolerance requirements at the 99, 90, and 50% probability levels: $\{YY_{v\_99\_spec}, YY_{v\_90\_spec}, YY_{v\_50\_spec}\}$ and/or $\{YY_{h\_99\_spec}, YY_{h\_90\_spec}, YY_{h\_50\_spec}\}$ and/or $\{YY_{r\_99\_spec}, YY_{r\_90\_spec}, YY_{r\_50\_spec}\}$ – values specified in % (specify one, two, or three sets of values per "and/or" of Equation (5.2.1-1).
  - Section 5.4 discusses recommended values in detail
  - In order to provide context, typical values of the above for a well-calibrated commercial satellite EO imaging system and numerous samples of error are on the order of $\{YY_{h\_99\_spec}, YY_{h\_90\_spec}, YY_{h\_50\_spec}\}=\{97, 85, 44\ \%\}$.
  - The normalized error tolerance requirements (e.g., $\{YY_{h\_99\_spec}, YY_{h\_90\_spec}, YY_{h\_50\_spec}\}$ are unrelated to the lub confidence-level YY for the validation of accuracy requirements – they just share some common symbology
- Description of operational constraints for applicable NSG geolocation system specific to the particular system in consideration.

Underlying Definitions and Computations:

- Normalized error (unit-less) for a specified level of probability XX (99, 90, or 50%) and a specific sample k corresponds to the sample radial error divided by the sample error's predicted magnitude or predicted "radial" at the XX probability level, computed from the corresponding covariance matrix (predicted accuracy) $C_{X\_}pred_{s\_k}$; the following are ellipsoidal-based normalized error samples, the default method for computation [7]:

  - $\epsilon v\_norm_{XX\_s\_k} = (\epsilon X_{s\_k}^T (C_{X\_}pred_{s\_k})^{-1} \epsilon X_{s\_k})^{1/2} / d_{v\_XX}$ (unit-less); (5.2.1-2)
    $\epsilon X_{s\_k} = \epsilon z_{s\_k}$ (m), $C_{X\_}pred_{s\_k}$ corresponding $1x1$ covariance matrix (m^2);

- $\epsilon h\_norm_{XX\_s\_k} = (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{1/2} / d_{h\_XX}$ (unit-less);
  $\epsilon X_{s\_k} = [\epsilon x_{s\_k} \quad \epsilon y_{s\_k}]^T$(m), $C_X\_pred_{s\_k}$ corresponding $2x2$ covariance matrix (m^2)
- $\epsilon r\_norm_{XX\_k} = (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{1/2} / d_{r\_XX}$ (unit-less);
  $\epsilon X_{s\_k} = [\epsilon x_{s\_k} \quad \epsilon y_{s\_k} \quad \epsilon z_{s\_k}]^T$(m), $C_X\_pred_{s\_k}$ corresponding $3x3$ covariance matrix (m^2).

- $d_{v\_99} = 2.576$, $d_{v\_90} = 1.645$, $d_{v\_50} = 0.674$ (unit-less) (5.2.1-3)
- $d_{h\_99} = 3.035$, $d_{h\_90} = 2.146$, $d_{h\_50} = 1.177$ (unit-less)
- $d_{r\_99} = 3.368$, $d_{r\_90} = 2.500$, $d_{r\_50} = 1.538$ (unit-less).

- Note that a normalized error sample at an XX% probability level can also be equivalently and explicitly written as the ratio of the (vertical, horizontal, or 3d) radial error divided by its predicted radial at the XX% probability level, illustrated as follows for horizontal normalized error:

  - $$\begin{aligned} \epsilon h\_norm_{XX\_s\_k} &= (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{1/2} / d_{h\_XX} \\ &= |\epsilon X_{s\_k}| / (d_{XX} |\epsilon X_{s\_k}| (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{-1/2}) \\ &= \epsilon h_{s\_k} / (d_{XX} \epsilon h_{s\_k} (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{-1/2}) \\ &\equiv \epsilon h_{s\_k} / \epsilon h\_pred\_radial_{XX\_s\_k} \end{aligned}$$ (5.2.1-4)

  - sample *k* horizontal (radial) error $\epsilon h_{s\_k} = |\epsilon X_{s\_k}| = \sqrt{\epsilon x_{s\_k}^2 + \epsilon y_{s\_k}^2}$
  - sample *k* horizontal predicted radial at an XX% probability level $\epsilon h\_pred\_radial_{XX\_s\_k}$ is equal to: $d_{XX} \epsilon h_{s\_k} (\epsilon X_{s\_k}^T (C_X\_pred_{s\_k})^{-1} \epsilon X_{s\_k})^{-1/2}$
  - Equation (5.2.1-4) and its sub-bullets follow from the general equation for the error ellipsoid: $\epsilon X^T C_X^{-1} \epsilon X = d^2$, where the value of d is parameterized by the dimension of ϵX and the desired probability-level (see TGD2a Section 5.3) $\epsilon h\_norm_{XX} \equiv \epsilon h / \epsilon h\_pred\_radial_{XX}$
  - example of horizontal error and its predicted 90% radial provided in Figure 5.2.1-1 below using green 90% error ellipse computed from predicted error covariance matrix:

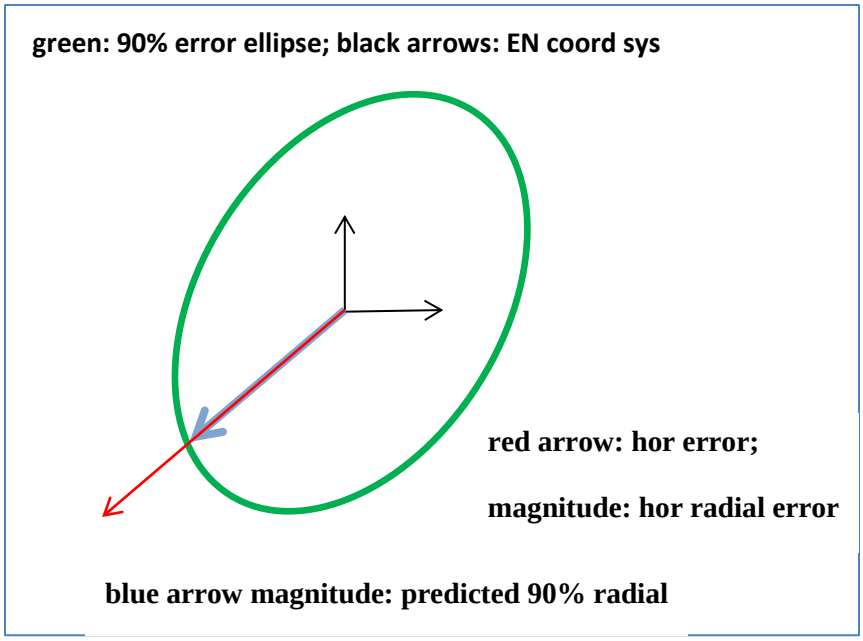


Figure 5.2.1-1: Example of horizontal error sample, its radial, and its corresponding 90% predicted radial

- $prob$ in Equation (5.2.1-1) is defined over all MIG-type extractions within operational constraints.

Recommended but optional additions:

- Minimum number of i.i.d. error samples for corresponding formal validation of the predicted accuracy specification – value specified as $n\_iid\_min\ \_pred\_acc\_val$ (unitless).
- Normalized error test at approximately 0.999999 probability (max none to exceed):
  - An optional fourth test for the applicable normalized errors, illustrated as follows for horizontal errors (similar tests for vertical and 3d errors): $\epsilon h\_norm_{999999\_s\_k} < 1$ for all samples, if not true for one or more samples, validation of predicted accuracy fails. This prevents extreme outliers. The value of $\epsilon h\_norm_{999999\_s\_k}$ is computed as specified in Equation (5.2.1-2) (or Equation (5.2.1-4)) using the approximate value $d_{h\_999999} = 5.3$ (similarly, $d_{v\_999999} = 5$ and $d_{r\_999999} = 5.5$). Corresponding probability corresponds to approximately less than "one in a million".

Other optional additions:

- Compute scalar accuracy metric-based normalized errors instead of ellipsoidal-based normalized errors – required action specified as $norm\_errors\_scalar\_acc\_based$ = "true":
- The scalar accuracy metric-based computation is directly analogous to the ellipsoidal-based computation of normalized error (Equation (5.2.1-4)), except that it utilizes the standard scalar accuracy metrics LE, CE, and SE at the appropriate probability level $XX\%$ to normalize ϵv, ϵh, and ϵr, respectively; for example, for normalized horizontal error:

- $\varepsilon h\_norm_{XX\_s\_k} = \epsilon h_{s\_k}/CEXX_{s\_k}$, (5.2.1-5)
  where $CEXX_{s\_k}$ is computed from the corresponding $2x2$ error covariance matrix $C_X\_pred_{s\_k}$ corresponding to sample k.
- Similarly, $\varepsilon v\_norm_{XX\_s\_k} = \epsilon v_{s\_k}/LEXX_{s\_k}$ and $\varepsilon r\_norm_{XX\_s\_k} = \epsilon r_{s\_k}/SEXX_{s\_k}$.
- The same normalized error tolerance requirements (e.g., $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$) and tests (Equation (5.2.1-1)) are applicable when normalized errors are computed using scalar accuracy metrics, although the actual tolerance values may differ from those used with predicted radials (see Section 5.4.3)
- See TGD 2a, Sections 5.4.2, 5.4.1, and 5.4.3, for the appropriate computation of $CEXX_{s\_k}$, $LEXX_{s\_k}$, and $SEXX_{s\_k}$, respectively, from $C_X pred_{s\_k}$.
  - If the normalized error test at 0.999999 probability level is an option as well, the following are applicable approximations: $CE999999 = \left(\frac{d_{h\_999999}}{d_{h\_90}}\right) CE90$, etc.

- The scalar accuracy metric-based method for normalization of errors and related validation are equally "correct" as the ellipsoidal-based method described earlier. However, the ellipsoidal-based method is preferred as it is used to validate error covariance matrices output from MIG-type extractions, not scalar accuracy metrics generated from these error covariance matrices which contain less inherent information (see TGD 1 for a corresponding discussion).

Comments:

- One reason to include normalized 3d errors in the specification (Equation (5.2.1-1)) is that they include the effects of correlation between vertical and horizontal components of error.
- The normalized error requirements of Equation (5.2.1-1) and corresponding computations are based on an assumed mean-zero multi-variate Gaussian (Normal) distribution of underlying errors (these are component errors $\epsilon x, \epsilon y, \epsilon z$, not computed radial errors $\epsilon v, \epsilon h, \epsilon r$):
  - In theory, if the assumed mean-zero multi-variate Gaussian distribution of errors was indeed applicable and if there were essentially an unlimited number of i.i.d. error samples, the specified values for the normalized error tolerance requirements would approach: $YY_{v\_99\_spec} \rightarrow 99, .. , YY_{r\_50\_spec} \rightarrow 50$.
  - An assumed and specific probability distribution of errors is required for the specification of predicted accuracy due to normalization of errors, but not required for the specification of accuracy (Section 5.1.1).
- In practice, in order to compensate for only an approximate mean-zero multi-variate Gaussian distribution of errors and only a reasonable number of i.i.d. error samples, a specified normalized error tolerance requirement at the XX probability level corresponds to a value less than $XX\%$.
  - For example, assuming 100 samples, $YY_{h\_90\_spec} = 83$ instead of $YY_{h\_90\_spec} = 90$. That is, the actual value specified dictates the amount of "approximation pad"; in this example, a pad of $90\% - 83\% = 7\%$.
  - Guidance regarding recommended specified values for $YY_{v\_99\_spec}$, .. , $YY_{r\_50\_spec}$ are provided in Section 5.4.2 of this document.

- Reasons for the practical specification of normalized error using three different probability levels (99, 90, and 50%) in Equation (5.2.1-1), and with an opposite inequality sign for the 50% probability, are discussed in Section 5.5 of this document.
- Regarding Equation (5.2.1-5), if the error covariance matrix $C_X\_pred_{s\_k}$ is not available to compute the various $CEXX_{s\_k}$, and only $CE90_{s\_k}$ is provided instead, only horizontal errors normalized at the 90% level can be rigorously computed:
    - As opposed to error ellipsoids, circular error does not have a fixed scale-factor between probability levels, and therefore, corresponding $CE99_{s\_k}$ and $CE50_{s\_k}$ cannot be rigorously computed from $CE90_{s\_k}$ alone. However, the other tests involving $CE99_{s\_k}$ and $CE50_{s\_k}$ can be performed if need be by using the scaling approximations $CE99_{s\_k} \cong (d_{h\_99}/d_{h\_90})CE90_{s\_k}$ and $CE50_{s\_k} \cong (d_{h\_50}/d_{h\_90})CE90_k$, as discussed in Section 5.4.3 of this document.

5.2.2 Validation of Specified Predicted Accuracy Requirements

General statement: The predicted accuracy requirements for NSG geolocation system xxxx shall be validated as follows:

Explicit form:

- Normalized vertical error and/or (5.2.2-1)
    - $prob\{\epsilon v\_norm_{99} \leq 1\} \geq 0.YY_{v\_99\_spec}$ validated if:
        - {% of $\epsilon v\_norm_{99\_s\_k}$ samples$\leq 1\} \geq YY_{v_99_spec}$
    - $prob\{\epsilon v\_norm_{90} \leq 1\} \geq 0.YY_{v\_90\_spec}$ validated if:
        - {% of $\epsilon v\_norm_{90\_s\_k}$ samples$\leq 1\} \geq YY_{v_90_spec}$
    - $prob\{\epsilon v\_norm_{50} > 1\} \geq 0.YY_{v\_50\_spec}$ validated if:
        - {% of $\epsilon v\_norm_{50\_s\_k}$ samples$> 1\} \geq YY_{v_50_spec}$
    - All three of the above tests must pass for validation
- Normalized horizontal error and/or
    - $prob\{\epsilon h\_norm_{99} \leq 1\} \geq 0.YY_{h\_99\_spec}$ validated if:
        - {% of $\epsilon h\_norm_{99\_s\_k}$ samples$\leq 1\} \geq YY_{h_99_spec}$
    - $prob\{\epsilon h\_norm_{90} \leq 1\} \geq 0.YY_{h\_90\_spec}$ validated if:
        - {% of $\epsilon h\_norm_{90\_s\_k}$ samples$\leq 1\} \geq YY_{h_90_spec}$
    - $prob\{\epsilon h\_norm_{50} > 1\} \geq 0.YY_{h\_50\_spec}$ validated if:
        - {% of $\epsilon h\_norm_{50\_s\_k}$ samples$> 1\} \geq YY_{h_50_spec}$
    - All three of the above tests must pass for validation

- Normalized 3d error
    - $prob\{\epsilon r\_norm_{99} \leq 1\} \geq 0.YY_{r\_99\_spec}$ validated if:
        - {% of $\epsilon r\_norm_{99\_s\_k}$ samples$\leq 1\} \geq YY_{r_99_spec}$
    - $prob\{\epsilon r\_norm_{90} \leq 1\} \geq 0.YY_{r\_90\_spec}$ validated if:
        - {% of $\epsilon r\_norm_{90\_s\_k}$ samples$\leq 1\} \geq YY_{r_90_spec}$
    - $prob\{\epsilon r\_norm_{50} > 1\} \geq 0.YY_{r\_50\_spec}$ validated if:
        - {% of $\epsilon r\_norm_{50\_s\_k}$ samples$> 1\} \geq YY_{r_50_spec}$
    - All three of the above tests must pass for validation

Values specified:

- Normalized error tolerance requirements at the 99, 90, and 50% probability levels: $\{YY_{v\_99\_spec}$, $YY_{v\_90\_spec}$, $YY_{v\_50\_spec}\}$ and/or $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$ and/or $\{YY_{r\_99\_spec}$, $YY_{r\_90\_spec}$, $YY_{r\_50\_spec}\}$ – values specified in % (specify one, two, or three sets of values per Equation (5.2.2-1) consistent with Predicted Accuracy Specification's Equation (5.2.1-1)).
- Optional: Minimum number of i.i.d. error samples for corresponding formal validation of the predicted accuracy specification – value specified as $n\_iid\_min\_pred\_acc\_val$ (unitless) per Predicted Accuracy Specification per Section 5.2.1.
- Optional: Compute scalar accuracy metric-based normalized errors instead of ellipsoidal-based normalized errors – required action specified as $norm\_errors\_scalar\_acc\_based$ = "true".

Underlying Definitions and Computations:

- The normalized error sample computations (e.g., $\epsilon h\_norm_{90\_s\_k}$) for the tests of Equation (5.2.2-1) are ellipsoidal-based unless specified otherwise by $norm\_errors\_scalar\_acc\_based$ = "true". Both forms of computation are defined in Predicted Accuracy, Section 5.2.1.
- If specified as such, there must be at least $n\_iid\_min\_pred\_acc\_val$ i.i.d. samples in order to perform the tests of Equation (5.2.2-1) for formal validation.
- In addition, if the optional normalized error test at approximately 0.99999 probability (max none to exceed) was specified per Section 5.2.1, all corresponding normalized errors must be less than 1 or validation fails.

Comments:

- The larger the number of i.i.d. error samples used, the better (higher confidence) the validation of prediction accuracy capabilities:
    - If the minimum number of i.i.d. error samples was not specified as part of the Predicted Accuracy Specification ($n\_iid\_min\_pred\_acc\_val$), guidance on a reasonable number of i.i.d. samples is provided in Section 5.4 of this document, as well as the effects of too few samples.

5.2.3 Examples of the Specification of Predicted Accuracy Requirements and Corresponding Validation

All specific numbers are for illustrative purposes only, and roughly approximate those applicable to geolocations using stereo images from a commercial EO imaging satellite for specificity.

5.2.3.1 Predicted Accuracy Specification Example 1

- Normalized 3d error
    - $prob\{\epsilon r\_norm_{99} \leq 1\} \geq 0.YY_{r\_99\_spec} = 0.94$ (5.2.3.1-1)
    - $prob\{\epsilon r\_norm_{90} \leq 1\} \geq 0.YY_{r\_90\_spec} = 0.82$
    - $prob\{\epsilon r\_norm_{50} > 1\} \geq 0.YY_{r\_50\_spec} = 0.37$

- The same operational constraints specified in the earlier Accuracy Specification example are applicable.
- The above set of $0.YY_{r\_99\_spec}$, .. , $0.YY_{r\_50\_spec}$ correspond to a minimum number of i.i.d. samples $n\_iid\_min\_pred\_acc\_val$ = 100.
- The baseline ellipsoidal-approach for the computation of normalized error during Validation is applicable. It is not over-written with the optional use of scalar accuracy metrics, i.e., the flag $norm\_errors\_scalar\_acc\_based$ = "false".

5.2.3.2 Corresponding Validation of Example 1: Case 1

Error samples $\epsilon X_{s\_k} = [\epsilon x_{s\_k} \quad \epsilon y_{s\_k} \quad \epsilon z_{s\_k}]^T$, $k = 1,..,100$, were first simulated corresponding to 100 i.i.d. samples of 3d geolocation error relative to ground truth (as opposed to using actual or "field" error samples in an actual validation). Their corresponding $3x3$ predicted error covariance matrices $C_X\_pred_{s\_k}$ were simulated as well. The underlying sample solutions and their error covariance matrices correspond to MIG-type extractions.

For this particular simulation, MIG-type extraction error covariance matrices were generated approximately 0.95^2 times their actual error covariance matrix counterparts, in order that predicted accuracies are somewhat optimistic. (This a scalar multiple of the entire error covariance matrix, i.e., each matrix element is multiplied by the scalar.) Independent and identically distributed (i.i.d.) error samples were simulated consistent with their corresponding actual error covariance matrices.

Validation of the predicted accuracy requirements then began by implementing Equation (5.2.1-2) to compute $\epsilon r\_norm_{XX\_k}$ for samples $k = 1,..,100$ for each of the three probability levels $XX = 99, 90, and\ 50\%$. Then, for each probability level, the percent of the normalized error samples that were less than or equal to 1.0 (greater than 1.0 for the 50% probability level) were computed corresponding to the specific tests of Equation (5.2.2-1) applicable to normalized 3d error. Results are presented in Table 5.2.3.2-1 below. (Note that Required Value(s) in the table correspond to {$YY_{r\_99\_spec}$, $YY_{r\_90\_spec}$, $YY_{r\_50\_spec}$ }.) All three one-sided tests passed (as required), so predicted accuracy was validated.

The same pseudo-code (Appendix F) that was used for the validation of accuracy example of Section 5.1.3-2 was used for the above validation of predicted accuracy example. This simulation code was also used for the additional predicted accuracy validation examples presented in upcoming Sections 5.2.3.3 and 5.2.3.5., with appropriate simulation parameter settings tailored to the particular example. For a given example, the simulation generated appropriate MIG-type errors and predicted error covariance matrices for each sample, computed the corresponding predicted scalar metrics or scalar accuracy metrics for normalization of the error samples, performed the validation tests (Equation (5.2.2-1)), presented and plotted results.

Table 5.2.3.2-1: Predicted Accuracy Validation Results for 3d (Radial) Error (Example 1, Case 1)

Probability level (%)	normalized error samples <= 1 (%)	Required Value (minimum)	pass/fail
99	98	94	pass
90	84	82	pass
Probability level	% normalized error samples > 1	Required Value (minimum)	pass/fail
50	54	37	pass

The results of these tests can also be illustrated graphically by plotting (non-normalized) 3d radial error samples versus the corresponding predicted radial at the $XX\%$ probability level (see Equation (5.2.1-4)). This also gives more insight into errors versus their predicted accuracies as a function of predicted accuracy (radial) magnitude.

Figures 5.2.3.2-1 through 5.2.3.2-3 present test results. The percent of 3d radial error samples below the 45 degree line correspond to the percent of normalized 3d error samples less than or equal to one for the probability levels $XX = 99\ and\ 90\%$ (Figures 5.2.3.2-1 and 5.2.3.2-2, respectively). The percent of 3d radial error samples above the 45 degree line correspond to the percent of normalized 3d error samples greater than one for the probability level $XX = 50\%$ (Figure 5.2.3.2-3).

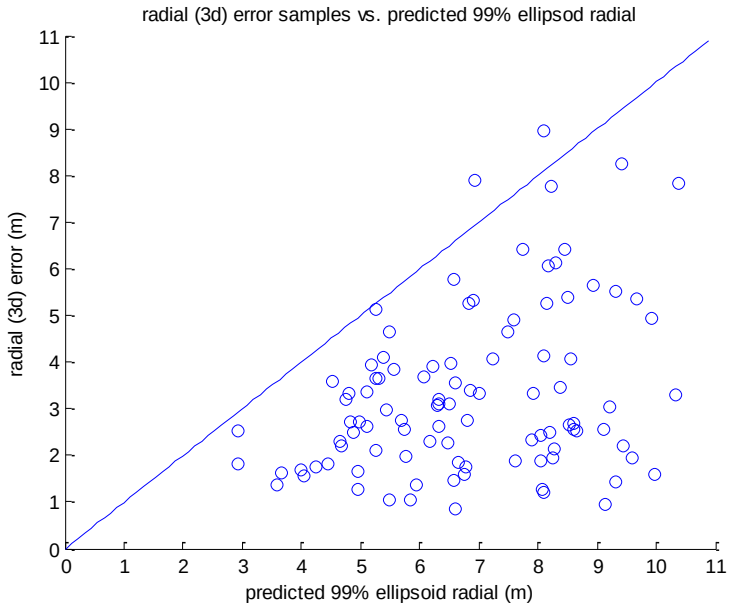


Figure 5.2.3.2-1: 3d radial error samples versus their corresponding predicted 99% radials – 98% below the 45 degree line

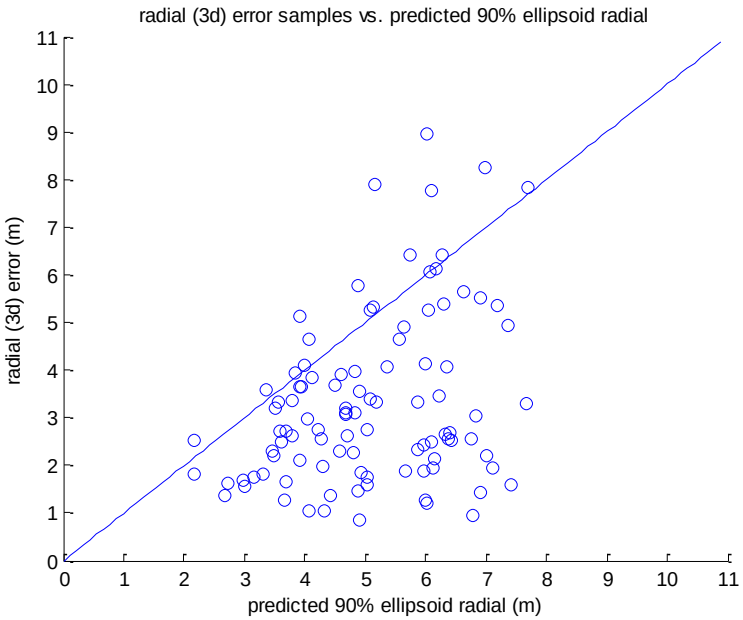


Figure 5.2.3.2-2: 3d radial error samples versus their corresponding predicted 90% radials – 84% below the 45 degree line

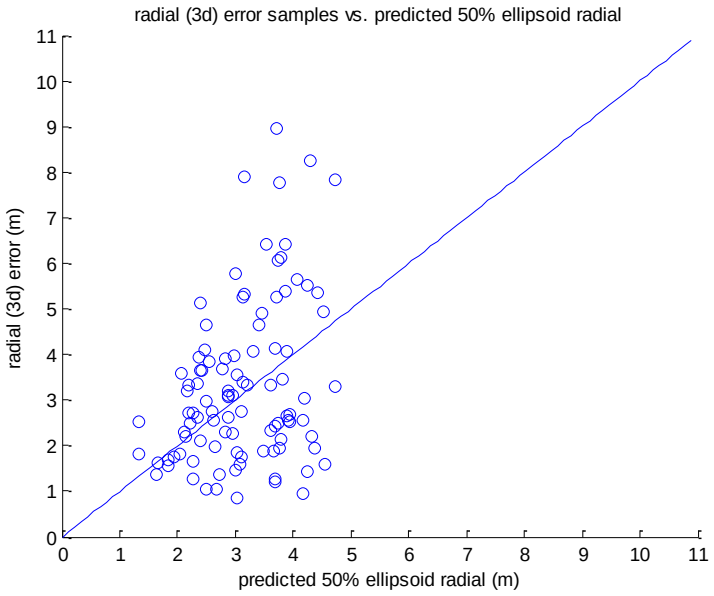


Figure 5.2.3.2-3: 3d radial error samples versus their corresponding predicted 50% radials – 54% above the 45 degree line

Because there is fixed scaling between predicted radials at different probability levels, we can also plot all three sets of results in one figure as radial error samples versus predicted 90% radials, as illustrated in Figure 5.2.3.2-4. The magenta line corresponds to the 99% validation test's (adjusted) 45 degree line,

the blue line to the 90% validation test's (unadjusted) 45 degree line, and the red line to the 50% validation test's (adjusted) 45 degree line. The slope of the adjusted 99% line is equal to (d_{r_99}/d_{r_90}), the slope of the unadjusted 45 degree line is equal to 1, and the slope of the adjusted 50% line is equal to (d_{r_50}/d_{r_90}). The number of error samples below the first two lines (magenta and blue) and the number of error samples above the last (red) line correspond to the results of the previous three figures as well as to Table 5.2.3.2-1.

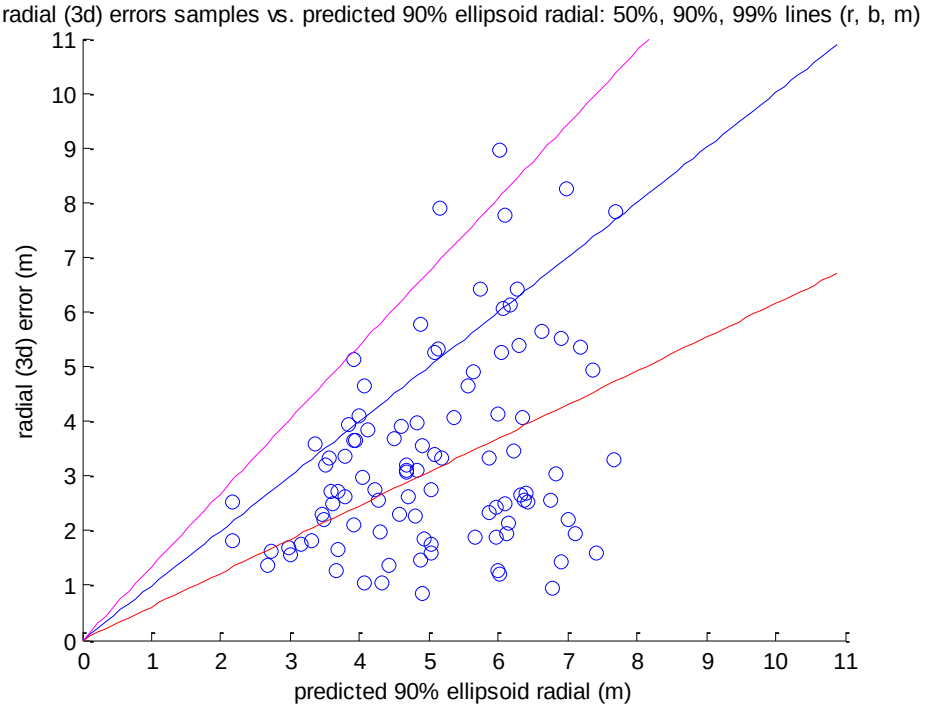


Figure 5.2.3.2-4: 3d radial error samples versus their corresponding predicted 90% radials with three validation test lines; Example 1-Case 1 results

Finally, recall that the optional normalized error test at approximately 0.999999 probability (max none to exceed) was not specified. If it had been, based on the above figure, it clearly would have passed as $d_{r\_999999}/d_{r\_99} \cong 1.63$, and no radial error sample is above 1.63 times its corresponding value on the 99% line (upper magenta line) in Figure 5.2.3.2-4. (The actual line corresponding to 99.9999% probability is typically not drawn in validation even when the optional test is specified.)

5.2.3.3 Corresponding Validation of Example 1: Case 2

This example is exactly like the previous examples except that the predicted error covariance was approximately only 0.80^2 times the actual error covariance matrix, i.e., significantly optimistic. The following presents the simulation results. Validation of predicted accuracy fails as it should, i.e., predicted accuracy was too optimistic per the set of normalized error tolerance requirements: $YY_{r\_99\_spec} = 94$, $YY_{r\_90\_spec} = 82$, and $YY_{r\_50\_spec} = 37$. Section 5.4 discusses the use of these specific values.

Table 5.2.3.3-1: Predicted Accuracy Validation Results for 3d (Radial) Error (Example 1, Case 2)

Probability level (%)	normalized error samples <= 1 (%)	Required Value (minimum)	pass/fail
99	95	94	pass
90	73	82	fail
Probability level	% normalized error samples > 1	Required Value (minimum)	pass/fail
50	70	37	pass

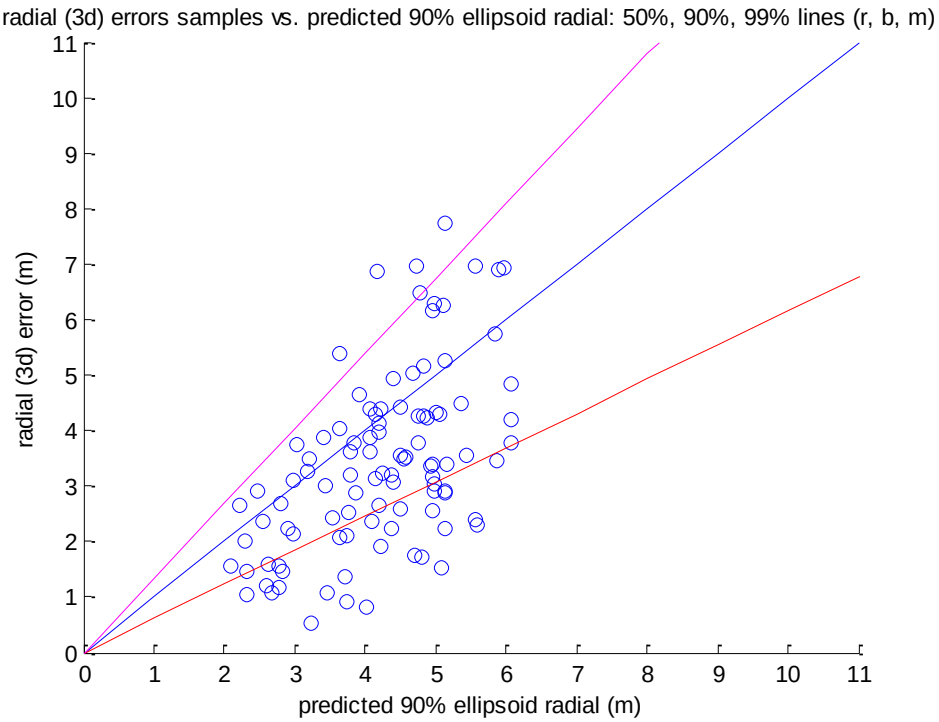


Figure 5.2.3.3-1: 3d radial error samples versus their corresponding predicted 90% radials with three validation test lines; Example 1-Case 2 results

5.2.3.4 Predicted Accuracy Specification Example 2

- Normalized vertical error and (5.2.3.4-1)
  - $prob\{\epsilon v\_norm_{99} \leq 1\} \geq 0.YY_{v\_99\_spec} = 0.95$
  - $prob\{\epsilon v\_norm_{90} \leq 1\} \geq 0.YY_{v\_90\_spec} = 0.83$
  - $prob\{\epsilon v\_norm_{50} > 1\} \geq 0.YY_{v\_50\_spec} = 0.40$
- Normalized horizontal error
  - $prob\{\epsilon h\_norm_{99} \leq 1\} \geq 0.YY_{h\_99\_spec} = 0.95$
  - $prob\{\epsilon h\_norm_{90} \leq 1\} \geq 0.YY_{h\_90\_spec} = 0.83$
  - $prob\{\epsilon h\_norm_{50} > 1\} \geq 0.YY_{h\_50\_spec} = 0.39$

- The same operational constraints specified in the above Accuracy Specification example are applicable.
- The above set of $0.YY_{v\_99\_spec}$, .. , $0.YY_{h\_50\_spec}$ correspond to a minimum number of i.i.d. samples $n\_iid\_min\_pred\_acc\_val$ = 100.
- The baseline ellipsoidal-approach for the computation of normalized error during Validation is not applicable. It is over-written with the optional use of scalar accuracy, i.e., the indicator (or parameter "flag") $norm\_errors\_scalar\_acc\_based$ = "true".

5.2.3.5 Corresponding Validation of Example 2

A similar simulation was performed as for Example 1. However, corresponding validation of predicted accuracy was based on normalized vertical and horizontal error samples computed using predicted $LEXX$ and $CEXX$, respectively, instead of normalized 3d error samples computed using predicted 3d radials (see Equation (5.2.1-5)). In addition, the predicted error covariance matrix was 1.03^2 times the actual error covariance matrix, i.e., slightly pessimistic.

Results are presented below in a manner similar to those for Example 1, except that there are two sets of results, corresponding to vertical errors and horizontal errors, instead of the previous one set of results corresponding to 3d errors. All tests pass. Also note that there is no combined figure for all three tests for either vertical error samples or horizontal error samples. This is because horizontal scalar accuracy metrics do not scale between probability levels. (Only vertical scalar accuracy metrics scale.)

The horizontal error (confidence) ellipses corresponding to simulated error covariance matrix samples were fairly non-elongated, i.e., a typical square root of the smallest to largest eigenvalue in an error covariance matrix was approximately 0.8. This is typical for commercial E0 satellite imagery. Also, the error covariance matrix samples alternate between one of two specific error covariance matrices, which correspond to only two different specific imaging scenarios for simplicity, both within the assumed operational constraints.

Table 5.2.3.5-1: Predicted Accuracy Validation Results for Vertical (Radial) Error (Example 2)

Probability level (%)	normalized error samples <= 1 (%)	Required Value (minimum)	pass/fail
99	100	95	pass
90	86	83	pass
Probability level	% normalized error samples > 1	Required Value (minimum)	pass/fail
50	52	40	pass

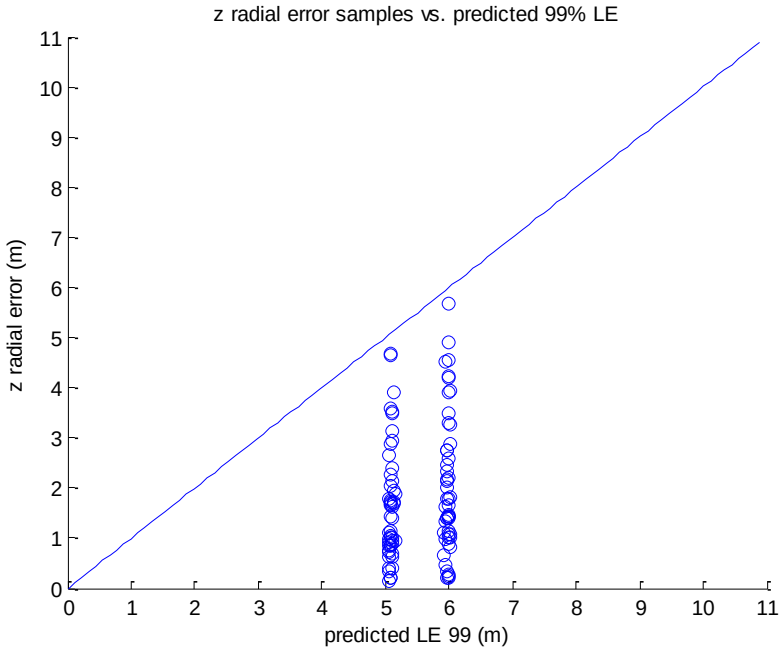


Figure 5.2.3.5-1: Vertical radial error samples versus their corresponding predicted LE99

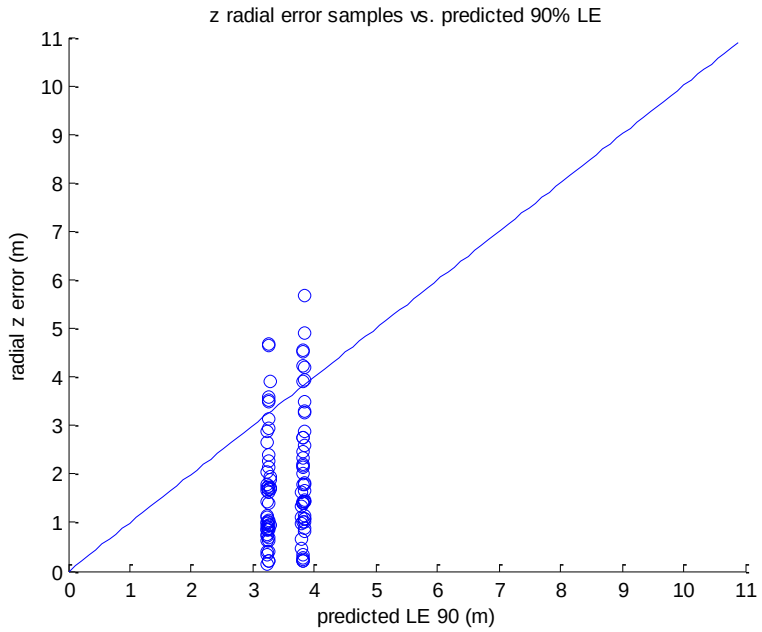


Figure 5.2.3.5-2: Vertical radial error samples versus their corresponding predicted LE90

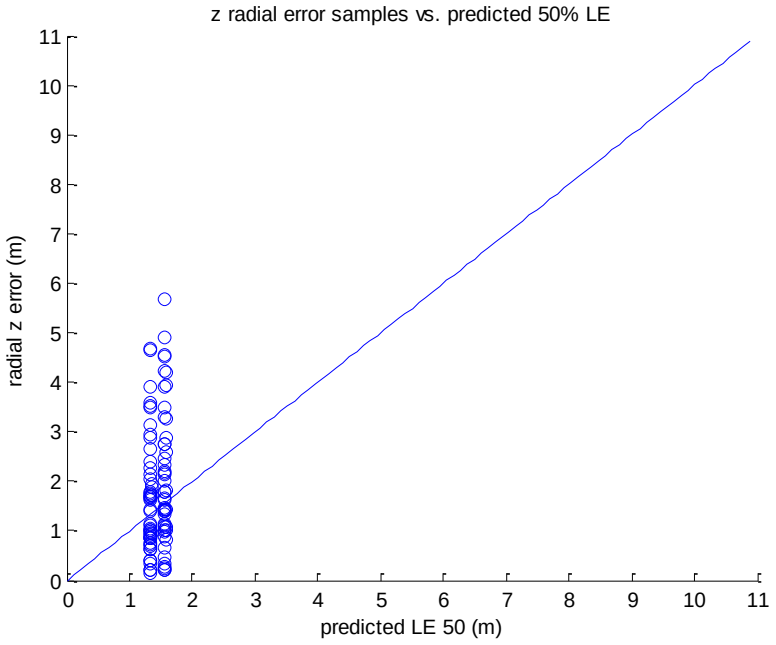


Figure 5.2.3.5-3: Vertical radial error samples versus their corresponding predicted LE50

Table 5.2.3.5-2: Predicted Accuracy Validation Results for Horizontal (Radial) Error (Example 2)

Probability level (%)	normalized error samples <= 1 (%)	Required Value (minimum)	pass/fail
99	96	95	pass
90	84	83	pass
Probability level	% normalized error samples > 1	Required Value (minimum)	pass/fail
50	51	39	pass

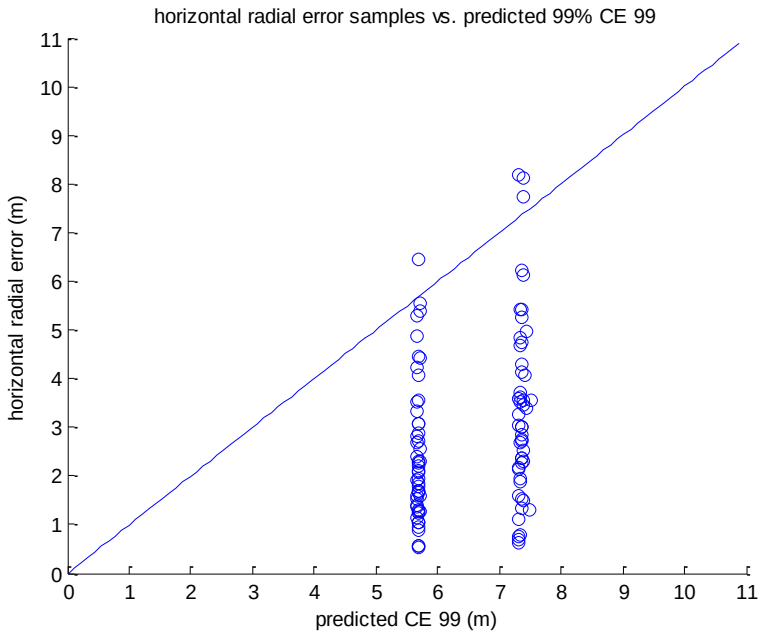


Figure 5.2.3.5-4: Horizontal radial error samples versus their corresponding predicted CE99

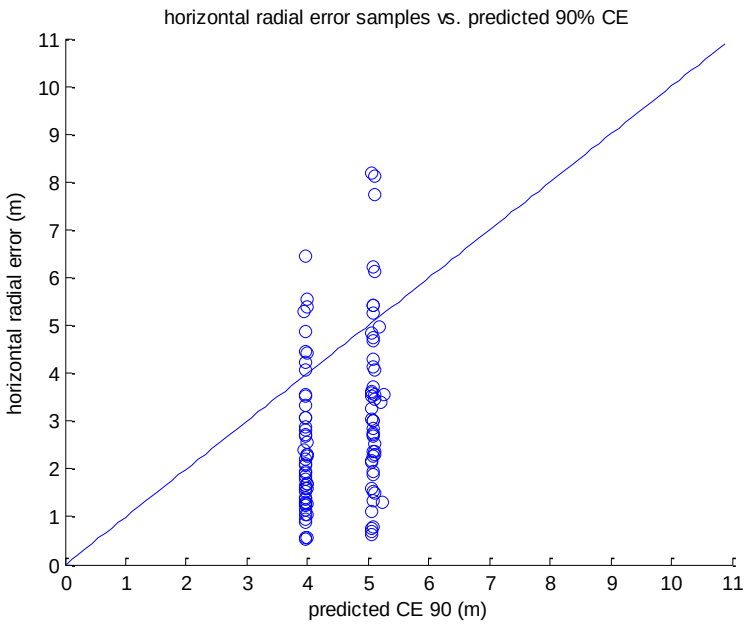


Figure 5.2.3.5-5: Horizontal radial error samples versus their corresponding predicted CE90

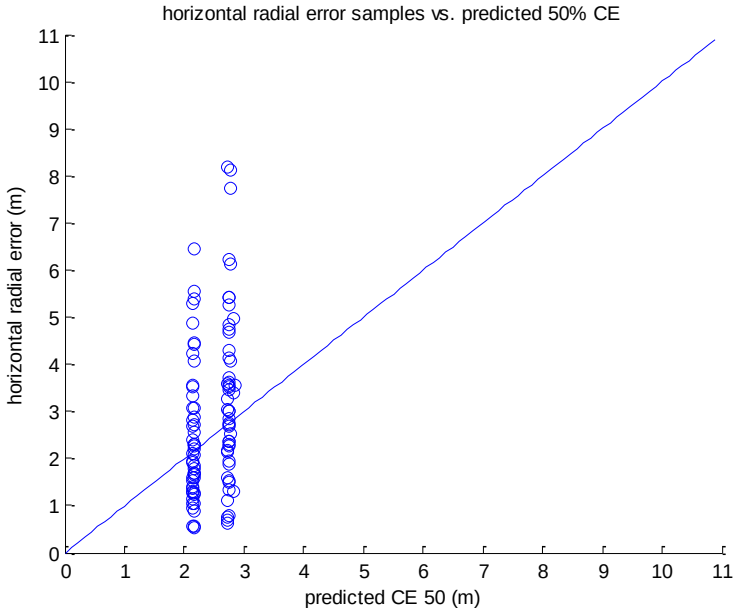


Figure 5.2.3.5-6: Horizontal radial error samples versus their corresponding predicted CE50

5.3 Relative Accuracy and Predicted Relative Accuracy Requirements and their Validation

The specifications for (absolute) accuracy and predicted (absolute) accuracy documented in Section 5.1 and Section 5.2, respectively, as well as their validation counterparts, are readily modified for the specification of relative accuracy and predicted relative accuracy between a pair of targets with the inclusion of distance-bin categories between the two geolocations, i.e., a set of requirements for each of multiple specified distance bins. An example of a typical set of distance bins in terms of nautical miles is as follows: $\{dist_1 < 1,\ 1 \leq dist_2 < 15\ ,\ 15 \leq dist_3 < 60,\ \ 60 \leq dist_4\}$. (Note: distances can also be specified in terms of other units, such as pixels, instead of nautical miles.)

For each specified distance bin, the corresponding relative accuracy and predicted relative accuracy requirements corresponding to a pair of targets (geolocations) are duplicate in form to those previously documented for (absolute) accuracy and predicted (absolute) accuracy in Sections 5.1 and 5.2, respectively. However, all corresponding specified errors and parameter values have "rel_bin_i", where $i = 1,..,n$ distance bins, added to their names, and associated processing is modified accordingly as detailed below. In addition, the corresponding MIG-type extractions for each target in a target-pair are assumed generated using the same sensor data. For example, if applicable sensor data corresponds to measurements of a target in a pair of stereo images, both targets are assumed extracted from that same pair of images.

5.3.1 Changes in Notation and Underlying Calculations for Relative Accuracy

As an example of the changes in notation for definitions and their underlying calculations from those corresponding to accuracy to those corresponding to relative accuracy, the following are applicable to horizontal (radial) errors and distance bin 2:

- $\epsilon h \rightarrow \epsilon h\_rel\_2$ and $CEXX_{spec} \rightarrow rel\_2\_CEXX_{spec}$. (5.3.1-1)
- $\epsilon h\_rel\_2 \equiv \sqrt{\left(\epsilon x_{s\_i} - \epsilon x_{s\_j}\right)^2 + \left(\epsilon y_{s\_i} - \epsilon y_{s\_j}\right)^2}$ where i and j correspond to sample errors corresponding to two different geolocations with a corresponding distance between them within distance bin 2; note that $\epsilon X_{s\_i} \equiv [\epsilon x_{s\_i} \quad \epsilon y_{s\_i}]^T$. (5.3.1-2)
- Typically, the value $rel\_2\_CEXX_{spec} < CEXX_{spec}$:
    - the difference between the two geolocation errors typically becomes smaller the smaller the distance bin due to positively correlated errors between the two targets, i.e., the statistical cancelation of common error;
    - however, for large distance bins, $rel\_2\_CEXX_{spec} \rightarrow \sqrt{2}CEXX_{spec}$, when errors between the two targets typically become uncorrelated.

5.3.2 Changes in Notation and Underlying Calculations for Predicted Relative Accuracy

As an example of the changes in notation for definitions and their underlying calculations from those corresponding to predicted accuracy to those corresponding to predicted relative accuracy, the following are applicable to horizontal (radial) errors and distance bin 2:

- $\epsilon h\_norm_{90} \rightarrow \epsilon h\_rel\_2\_norm_{90}$ and $0.YY_{h\_90\_spec} \rightarrow rel\_2\_0.YY_{h\_90\_spec}$. (5.3.2-1)
- $\epsilon h\_rel\_2\_norm_{XX\_s\_k} = (\epsilon Xrel_{s\_k}^{T}(rel\_C_X\_pred_{s\_k})^{-1}\epsilon Xrel_{s\_k})^{1/2}/d_{h\_XX}$ (unit-less): (5.3.2-2)
  - $\epsilon Xrel_{s\_k} \equiv \epsilon X_{s\_ki} - \epsilon X_{s\_kj}$, corresponding to unique sample pair k involving error samples for geolocations i and j within distance bin 2. (5.3.2-3)
  - $rel\_C_X\_pred_{s\_k} \equiv C_{X\_pred_{s\_ki}} + C_{X\_pred_{s\_kj}} - C_{X\_pred_{s\_kij}} - C_{X\_pred_{s\_kji}}$: (5.3.2-4)
    - $rel\_C_X\_pred_{s\_k}$ is the predicted relative error covariance matrix for relative error sample (sample pair) k
    - $C_{X\_pred_{s\_ki}}$ is the predicted error covariance matrix for geolocation i and $C_{X\_pred_{s\_kij}}$ is the predicted error cross-covariance matrix between geolocations i and j.
- If the non-default use of scalar accuracy metrics were specified for normalization of error for the predicted accuracy requirement, they are also specified for the normalization of relative error for the predicted relative accuracy requirement. For example:
  - $\epsilon h\_rel\_2\_norm_{XX\_s\_k} = \epsilon h\_rel\_2_k/rel\_CEXX_{s\_k}$, (5.3.2-5)
    where $rel\_CEXX_{s\_k}$ is computed from the $2x2$ error covariance matrix $rel\_C_X\_pred_{s\_k}$ corresponding to the relative error sample k.
- Typically $rel\_2\_0.YY_{h\_90\_spec} \cong 0.YY_{h\_90\_spec}$, which is also typical for all other distance bins:
  - The assumption that errors are mean-zero multi-variate Gaussian distributed implies the same general characteristics for relative errors due to the properties of the multi-variate Gaussian probability distribution; thus the "pad" for a given probability distribution for normalized errors should remain approximately the same for normalized relative errors.

5.3.3 Other Parameter Changes for Relative Accuracy and Predicted Relative Accuracy

Other applicable parameters involve the minimum number of i.i.d. error samples and the required confidence level. The values of the minimum number of i.i.d. error samples for relative accuracy and predicted relative accuracy are specified for each distance bin. Independent samples dictate that, within a given distance bin (but not across bins), one error sample can be associated with one and only one sample-pair. Specification of the minimum number of i.i.d. error samples per distance bin also accounts for typical distributions of samples: more in the "middle" distance bins and fewer in distance bins corresponding to either very small or very large distances.

Validation processing for relative accuracy and predicted relative accuracy is virtually the same as for their absolute accuracy and predicted accuracy counterparts, other than the change in notation and processing described above. Also, validation processing is performed for each specified distance bin.

The above general descriptions can seem somewhat complicated due to "layers" of unavoidable notation, but actual relative accuracy and predicted relative accuracy requirements (and corresponding validation) are straightforward as illustrated by the following examples. Only two distance bins are

specified in the examples for simplicity, and underlying definitions (already covered in the above subsections) are not included. Again, all specific numbers are for illustrative purposes only, and roughly approximate those applicable to geolocations extracted from single images from a commercial EO imaging satellite, i.e., monoscopic target extraction. Note that in order to use independent error samples, corresponding images used for validation should not be same-pass images to minimize correlated sensor support data errors between the images. Also, only one pair of MIG extractions should be used per distance bin per image – more on this later in Section 5.6.

In addition, error samples are generated using the corresponding ground truth's elevation value as the external elevation source in the MIG-type monoscopic target extraction. Thus, specified relative horizontal accuracy and predicted relative horizontal accuracy are virtually independent of elevation error. (Neither relative vertical or 3d accuracies or predicted relative vertical or 3d accuracies are applicable.)

Subsection 5.3.4 presents an example of the specification of relative accuracy requirements, and Subsection 5.3.5 presents an example of the specification of predicted relative accuracy requirements. (Examples of corresponding validation are not included as they are straight-forward extensions of those corresponding to accuracy and predicted accuracy requirements.)

5.3.4 Relative Accuracy Specification Example

- Two distance bins in nautical miles: $bin\_1 < 10$ and $10 \leq bin\_2$

- $prob\{\epsilon h\_rel\_1 \leq rel\_1\_CE90_{spec} = 2.5\ meters\} \geq 0.90$ (5.3.4-1)
- $prob\{\epsilon h\_rel\_2 \leq rel\_2\_CE90_{spec} = 4\ meters\} \geq 0.90$

    - Note that the equivalent alternate (non-percentile) form for specification was used above; see Equation (5.1.1-3) for their (absolute) accuracy specification counterparts.

- Operational constraints: Above requirements applicable to MIG monoscopic extractions using corresponding imaging geometry constraints: max off-nadir angle=…etc. Extraction mensuration errors assumed no larger than 1 pixel (one-sigma) in each of the image line and sample directions for each target.
- Validation of the above should be performed at a $conf\_acc\_val\_rel = 90$ % confidence level and with a minimum number of individual i.i.d. error samples $n\_iid\_min\_acc\_val\_rel\_1$ = 75 and $n\_iid\_min\_acc\_val\_rel\_2$ =50 required.

5.3.5 Predicted Relative Accuracy Specification Example

- normalized relative error bin_1 (5.3.5-1)
  - $prob\{\epsilon h\_rel\_1\_norm_{99} \leq 1\} \geq rel\_1\_0.YY_{h\_99\_spec} = 0.94$
  - $prob\{\epsilon h\_rel\_1\_norm_{90} \leq 1\} \geq rel\_1\_0.YY_{h\_90\_spec} = 0.82$
  - $prob\{\epsilon h\_rel\_1\_norm_{50} > 1\} \geq rel\_1\_0.YY_{h\_50\_spec} = 0.37$
- normalized relative error bin_2
  - $prob\{\epsilon h\_rel\_2\_norm_{99} \leq 1\} \geq rel\_2\_0.YY_{h\_99\_spec} = 0.93$
  - $prob\{\epsilon h\_rel\_2\_norm_{90} \leq 1\} \geq rel\_2\_0.YY_{h\_90\_spec} = 0.80$
  - $prob\{\epsilon h\_rel\_2\_norm_{50} > 1\} \geq rel\_2\_0.YY_{h\_50\_spec} = 0.35$

- The same operational constraints specified in the above Relative Accuracy Specification example are applicable.
- The above sets of normalized error tolerance requirements $\{rel\_1\_0.YY_{h\_99\_spec}$, .. ,*rel_1_* $0.YY_{h\_50\_spec}\}$ and $\{rel\_2\_0.YY_{h\_99\_spec}$, .. ,*rel_2_* $0.YY_{h\_50\_spec}\}$ correspond to the use of all possible independent pairs of error samples in each distance bin. Note that the number of pairs is expected to be somewhat small per distance bin, hence lower values for the above sets relative to typical (absolute) predicted accuracy counterparts to approximately account for less statistical significance.

5.4 Predicted Accuracy Requirements and their Validation Sensitivities to Sample Size and Predicted Accuracy Fidelity

This section addresses recommended values for the normalized error tolerance requirements at the 99, 90, and 50% probability levels in support of the specification and validation of (absolute) predicted accuracy requirements (Section 5.2). Specifically, recommended values for:

- $\{YY_{v\_99\_spec}$, $YY_{v\_90\_spec}$, $YY_{v\_50\_spec}\}$ and/or $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$ and/or $\{YY_{r\_99\_spec}$, $YY_{r\_90\_spec}$, $YY_{r\_50\_spec}\}$ – values specified in % (specify one, two, or three sets of values per Equation (5.2.1-1).

Section 5.4.1 provides an overview on sensitivities, i.e., what affects reasonable values for the above $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$, etc. Section 5.4.2 describes specific recommended values and their consequences. Section 5.4.3 describes complications and recommended values when scalar accuracy metrics (e.g. $CE90$) are used instead of predicted radials for the normalization of error samples. Section 5.4.4 discusses associated recommended research.

Examples presented in Sections 5.4.1, 5.4.2, and 5.4.3 correspond to the specification and validation of predicted horizontal accuracy (horizontal errors) only for ease of illustration.

5.4.1 Overview of Sensitivities

Appropriate values for normalized error tolerance requirements are sensitive to both the desired level of predicted accuracy fidelity and the number of samples available to validation (statistical significance). The former essentially corresponds to the fidelity of the predictive statistical error model(s) for the corresponding NSG systems. In particular, the fidelity of the resultant predicted error covariance matrix

output from MIG-type extractions of target geolocations relative to the true error covariance matrix. Although the true error covariance matrix is not known explicitly, actual error samples are available in the validation process. Also, the predicted error covariance matrices vary somewhat over the error samples, as each MIG-type extraction corresponds to a slightly different "operating point" within the specified operational scenario. It is expected that the appropriate level of predicted accuracy fidelity is NSG system-specific.

The following tables illustrate the above individual effects assuming horizontal errors and "ideal" and independent conditions, i.e. the effects of the number of samples only, and the effects of predicted accuracy fidelity only. Monte-Carlo simulations were performed with i.i.d. error samples and corresponding predicted error covariance matrices generated for the specified number of samples per realization. The error samples were generated consistent with the true error covariance matrix for each sample.

First we illustrate the effects of the number of samples or statistical significance on normalized horizontal errors; perfect predicted accuracy fidelity is assumed.

Table 5.4.1-1: Percent of normalized errors <1 (or >1 if 50%-level test) vs. number of samples per realization; perfect predicted accuracy fidelity (sigma deviation = 0%), 500 realizations; 90% confidence.

% norm errors < 1	number of i.i.d. samples									
for norm levels (%):	25	50	100	150	200	250	300	500	1000	>>1000
99	96	96	98	98	98	98	98.3	98.4	98.6	99
90	80	84	86	86.6	87.5	87.6	87.6	88.2	88.7	90
50 (>1)	36	40	43	44.6	46	46	46.3	47.2	48.1	50

Thus, for example, with 90% confidence and assuming 50 i.i.d. error samples, we expect that at least 84% of horizontal errors normalized at the 90% probability level will be less than 1. If we set $YY_{h\_90\_spec}$ to the values 84, we expect successful validation of the corresponding test with a confidence of 90%, i.e., the horizontal normalized error test at the 90% probability level of Equation (5.2.2-1) will pass. Again, this also assumes perfect predicted accuracy fidelity.

Next we illustrate the effects of predicted accuracy fidelity on normalized horizontal errors. Many samples (1000) are applicable for each realization, thus, statistical significance due to a finite number of samples is only a secondary and minor factor. For this particular illustration, predicted accuracy fidelity is expressed using **"sigma deviation"** or sig_dev**,** the latter defined as the predicted error covariance matrix equal to $(1 + sig\_dev)^2$ times the actual error covariance matrix (applicable to each element of the covariance matrix). That is, the predicted error covariance matrix is a scalar multiple of the actual or true error covariance matrix. Thus, a negative-value for sigma deviation (sig_dev) corresponds to an example of optimistic predicted accuracy and a positive-value corresponds to pessimistic (conservative) predicted accuracy. In particular, a value of sigma devation (expressed as a percent) of -10% corresponds to each error component's predicted standard deviation (or "sigma") to be only $\sqrt{(1 + sig\_dev)^2} = (1 + sig\_dev) = 0.9$ or 90% of the true standard deviation of error.

Table 5.4.1-2: Percent of normalized errors <1 (or >1 of 50%-level test) vs. predicted accuracy fidelity expressed as sigma deviation = +/- xx %; 1000 samples per realization; 500 realizations; 90% confidence

% norm errors < 1	sigma deviaton (%)						
for norm levels (%):	-50	-30	-10	0	10	30	50
99	66.7	88.4	97	98.6	99.4	99.9	100
90	41.9	65.7	83.2	88.9	92.8	97.4	99.1
50 (>1)	82.6	69.2	55.1	48	41.1	29.2	19.4

Thus, for example, with 90% confidence and assuming a sigma deviation of -30%, we expect that at least 88.4% of horizontal errors normalized at the 99% probability level will be less than 1.

(The difference between Table 5.4.1-2's sigma deviation = 0% results and Table 5.4.1-1's corresponding 1000 i.i.d. sample results are less than or equal to 0.2% and due to different Monte-Carlo runs.)

5.4.2 Normalized Error Tolerance Values

The specified values for the normalized error tolerances, e.g. $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}\}$ of Equation (5.2.1-1) , are "tailored" to both an assumed minimum number of i.i.d. error samples available to the corresponding validation process, and to a required "level of fidelity" of the underlying predicted statistical error model, i.e., the fidelity of predicted accuracy, or more specifically, the predicted error covariance matrix. This process and resultant specified values are described in this section. The validation of predicted horizontal accuracy is assumed for convenience, followed by tables of baseline normalized error tolearance values for the validation of predicted horizontal, vertical, and 3d accuracies, respectively.

As detailed earlier in Section 4.2, the level (category) of predicted accuracy fidelity is defined in terms of a sigma deviation range as follows:

Table 5.4.2-1: Predicted accuracy fidelity categories versus sigma deviation range

"sigma deviation" (%) per pred acc fidelity category		
high	medium	low
-5 to +5	-15 to +20	-30 to +40

(The above table and Equation (5.4.2-1) are applicable to predicted vertical, horizontal, and 3d accuracies – the size of the error covariance matrices in the equation are 1x1, 2x2, and 3x3, respectively.)

Correspondingly, and assuming predicted horizontal accuracy, the $2x2$ predicted error covariance matrix (C_X_pred) is assumed to satisfy the following relationship with the $2x2$ true error covariance matrix (C_X_true):

$$(1 + sig\_dev_l)^2 C_X\_true \leq C_X\_pred \leq (1 + sig\_dev_r)^2 C_X\_true, \qquad (5.4.2\text{-}1)$$

where sig_dev_l and sig_dev_r correspond to the left and right end points, respectively, of the sigma deviation range (interval) for the desired category of predicted accuracy fidelity per Table 5.4.2-1. The scalar multiplier (e.g. $(1 + sig\_dev_l)^2$) multiplies each component of C_X_true to yield a corresponding scaled true error covariance matrix. (See Section 5.3.5 of TGD2a (predictive statistics) for the definition of $A \leq B$ and $A < B$ for two covariance matrices.)

The relationship between the error covariance matrices specified in Equation (5.4.2-1) is very general. In particular, C_X_pred need not be an explicit scalar multiple of C_X_true, i.e., need not have the same shape and/or orientation. However, it is "bounded" by scalar multiples of C_X_true. This is illustrated in Figure 5.4.2-1 below corresponding to medium predicted accuracy fidelity, and a repeat of Figure 4.2-2 for convenience. (See Section 5.3.4 of TGD 2a (predictive statistics) for the equivalence of error covariance matrices and corresponding probability ellipses.)

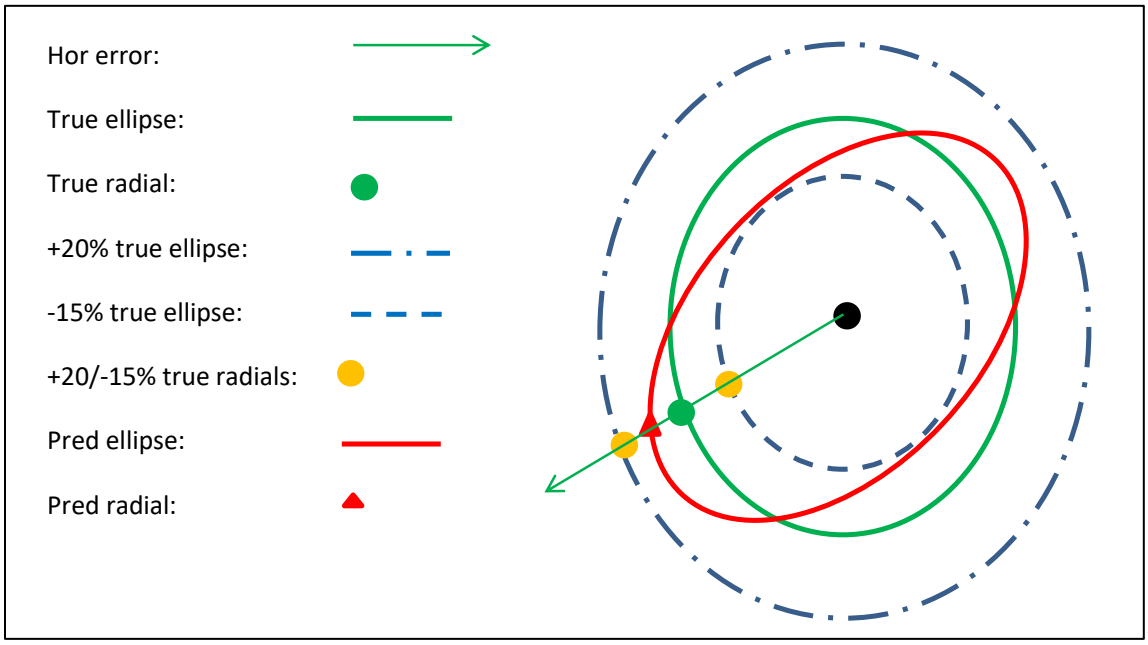


Figure 5.4.2-1: The predicted error covariance matrix C_X_pred (red probability ellipse) is bounded by scalar multiples of C_X_true (blue dot-dash and blue dash-dash probability ellipses)

Also, as seen in the above figure for an arbitrary sample of horizontal error (green arrow), the predicted radial (red triangle) computed using C_X_pred is always bounded by radials (gold dots) corresponding to the scaled true error covariance matrices $(1 - 0.15)^2 C_X\_true$ and $(1 + 0.20)^2 C_X\_true$. Of course, the actual values of these bounds (gold dots) are unknown as C_X_true is unknown.

The above bounds "enable" the use of normalized error tolerances for the validation of predicted horizontal accuracy. The bounds affect the differences in the sample-based probability distribution of normalized errors (sample radial error divided by predicted radial) relative to the theoretically correct probability distribution of normalized errors (sample radial error divided by true radial, assuming a very large number of samples). The normalized error tolerances and corresponding validation tests place limits on the differences between these two probability distributions at three different levels of probability.

Although not explicitly shown in Figure 5.4.2-1, the predicted radial is also equal to (not simply bounded by) a radial corresponding to a scaled true error covariance matrix $(1 + sig\_dev)^2 C_X\_true$, where $sig\_dev_l \leq sig\_dev \leq sig\_dev_r$, i.e., sig_dev is within the sigma deviation range for medium predicted accuracy per Table 5.4.2-1.

(The above characteristics are independent of the specific probability level $XX\%$ that is common to the probability ellipses in the figure and also applicable to the predicted radial $\epsilon h\_pred\_radial_{XX}$ computed via Equation (5.2.1-4).)

Figure 5.4.2-1 also assumed that Equation (5.4.2-1) was satisfied, and as such validation of medium predicted accuracy should pass as will be detailed later. If instead, $(1 + sig\_dev_r)^2 C_X\_true < C_X\_pred$, the red ellipse would completely encompass the dot-dash blue ellipse and validation should fail. Or, alternatively, if $C_X pred < (1 + sig\_dev_l)^2 C_X\_true$, the red ellipse would be completely encompassed by the dot-dot blue ellipse and validation should fail as well. See Appendix D.3 for further discussion and examples.

Note: the sigma deviation range of -30% to +40% for low predicted accuracy fidelity as defined in Table 4.2-1 is purposely somewhat non-symmetric and favors conservative error propagation, i.e., assumes that the underling error models for extraction will be somewhat conservative in order to compensate for the lack of error modeling "information". A similar non-symmetry is also true for medium predicted accuracy fidelity.

Normalized error tolerance values vs. predicted accuracy fidelity

The degree of maturity, level of system calibration, and the variability of the operational range of an NSG system affects the specified level of fidelity (Table 5.4.2-1) which is implicit in the specified values $\{YY_{h\_99\_spec}, YY_{h\_90\_spec}, YY_{h\_50\_spec}\}$ for flexibility. These specified values are detailed below:

The following tables present baseline normalized error tolerance requirements for the three probability levels (99, 90, and 50%) as a function of both the number of samples available for the tests (statistical significance), and the applicable level of predicted accuracy fidelity. The tables' tolerance values reflect the combined (and inter-related) effects of sample size and predicted accuracy fidelity level on the passing of all three normalized error tests simultaneously, i.e., those tests corresponding to 99%, 90%, and 50% probability levels. Tables 5.4.2-2 through 5.4.2-4 below correspond to normalized horizontal, vertical, and 3d errors, respectively. Appendix D details table generation and includes MATLAB pseudo-code.

Table 5.4.2-2: Normalized horizontal error tolerance requirements vs. number of i.i.d. samples; categorized by desired predicted accuracy fidelity

normalized test level	400 samples	100 samples	50 samples	25 samples	pred acc fidelity
99	97	95	93	90	
90	85	83	78	76	high
50	44	39	38	34	
99	95	90	88	84	
90	78	76	72	68	medium
50	36	30	30	24	
99	85	84	82	76	
90	65	64	60	54	low
50	25	24	18	14	

Table 5.4.2-3: Normalized vertical error tolerance requirements vs. number of i.i.d. samples; categorized by specified predicted accuracy fidelity (slightly larger than for horizontal normalized error counterparts)

normalized test level	400 samples	100 samples	50 samples	25 samples	pred acc fidelity
99	97	95	94	90	
90	86	83	80	76	high
50	44	40	38	34	
99	96	91	90	86	
90	80	78	74	70	medium
50	40	34	32	30	
99	88	86	84	80	
90	70	66	64	62	low
50	30	28	26	22	

Table 5.4.2-4: Normalized 3d error tolerance requirements vs. number of i.i.d. samples; categorized by specified predicted accuracy fidelity (slightly smaller than for horizontal normalized error counterparts)

normalized test level	400 samples	100 samples	50 samples	25 samples	pred acc fidelity
99	96	94	93	87	
90	84	82	78	73	high
50	42	37	37	31	
99	89	85	84	82	
90	72	71	69	66	medium
50	32	27	27	22	
99	80	79	79	72	
90	59	57	57	50	low
50	19	15	15	10	

The tolerances of the above tables were derived such that the probability of success was at least 90% within the appropriate "sigma deviation" range, and dropped-off as soon as possible outside of this range as illustrated by the following plots.

The effect of normalized error test tolerance values (categorized by predicted accuracy fidelity) on validation

The following plots (Figures 5.4.2-2 through 5.4.2-13) correspond to normalized horizontal error tests based on the use of the normalized error tolerance requirements contained in Table 5.4.2-2. They graphically illustrate the probability of success (confidence) of passing all three probability-level tests (Equation (5.2.2-1)) versus sigma deviation, and hence, passing validation.

The probability of success is reflected as the % of realizations, each containing the specified number of i.i.d. samples, which pass all three probability-level tests. Five hundred realizations were performed per sigma deviation value via a Monte-Carlo simulation as detailed in Appendix D. Also, the horizontal magenta-colored dotted lines in these plots correspond to confidence at the 95%, 90%, and 50% levels and are there for convenient reference; they are not directly related to the normalized error test tolerance probability-levels of 99%, 90%, and 50%.

Figure 5.4.2-2 is presented first and corresponds to high predicted accuracy fidelity and 400 i.i.d. samples. It is followed by interpretation of plot results, which generalize to the plots that follow that correspond to different combinations of predicted accuracy fidelity and number of samples.

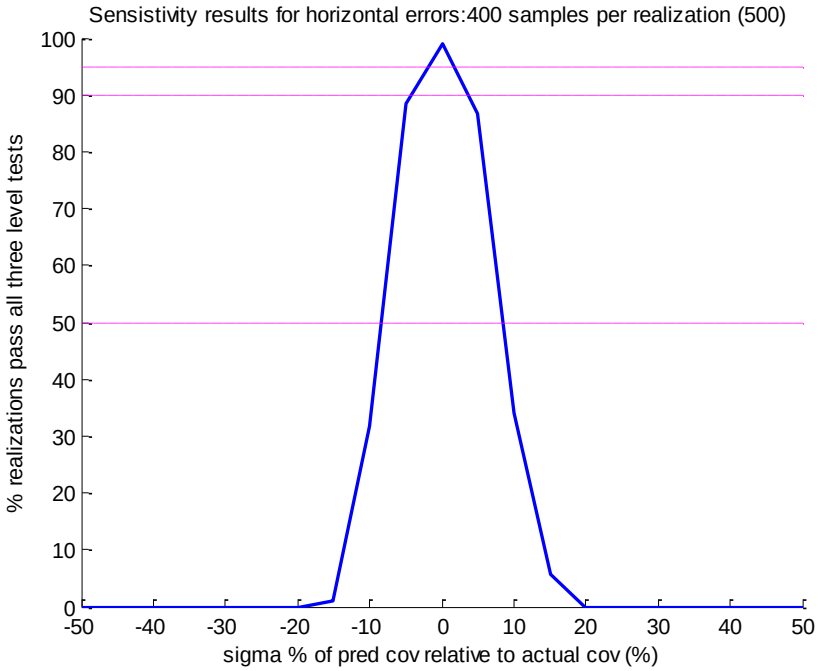


Figure 5.4.2-2: Probability (confidence) of passing all three normalized error tests; high fidelity, 400 samples

The plot's x-axis is titled "sigma % of pred cov relative to the actual cov" and corresponds to sigma deviation or sig_dev (%). The corresponding value on the y-axis is the probability of success (confidence) in passing validation if $C_X\_pred = (1 + sig\_dev)^2 C_X\_true$. The overall probability of validation success is the minimum y-value taken over the sigma deviation range $(sig\_dev_l, sig\_dev_r) = (-0.05, 0.05)$ on the x-axis, approximately equal to 0.92 in the above figure, and applicable to any predicted error covariance matrix that satisfies $(1 - 0.05)^2 C_X\_true \leq C_X\_pred \leq (1 + 0.05)^2 C_X\_true$.

(As explained earlier in the discussion regarding Figure 5.4.2-1, the relationship $(1 - 0.05)^2 C_X\_true \leq C_X\_pred \leq (1 + 0.05)^2 C_X\_true$ also places bounds on the value of the predicted radial relative to the true (actual) radial, or correspondingly, bounds on the error in the predicted radial's computed value which directly affects the validation tests via Equations (5.2.1-1) and (5.2.1-4).)

Type I validation errors correspond to failing at least one of the three probability-level tests when all should have passed within the corresponding sigma deviation range. In the above plot, the probability of a Type I error is less than 1 - 0.92 = 0.08 or 8%. Type II errors correspond to passing all three probability-level tests when at least one should have failed. In the above plot, the probability of a Type II error is less than 0.05 or 5% for any sigma deviation outside of the interval (-0.10, 0.10), and applicable to any predicted error covariance matrix that satisifies either $(1 + 0.10)^2 C_X\_true < C_X\_pred$ or $C_X pred < (1 - 0.10)^2 C_X\_true$.

The results of Figure 5.4.2-2 are as desired and dramatic per the fast "roll-off" of confidence outside of the desired sigma deviation range for high predicted accuracy fidelity. This is primarily due to a large number of samples (400). Other figures below show less dramatic roll-off with a lessor number of samples, as expected. In all cases (figures) the three probability-level normalized error tests work "in concert" to limit high confidence validation to within the desired sigma deviation range per the desired level of predicted accuracy fidelity (see Section 5.5 for a related discussion).

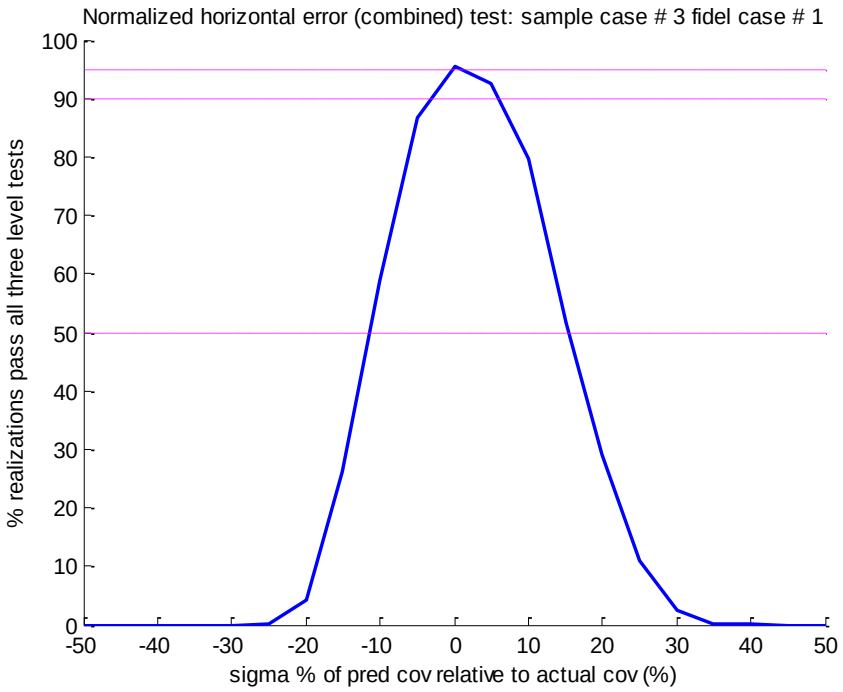


Figure 5.4.2-3: Probability of passing all three normalized error tests; high fidelity, 100 samples

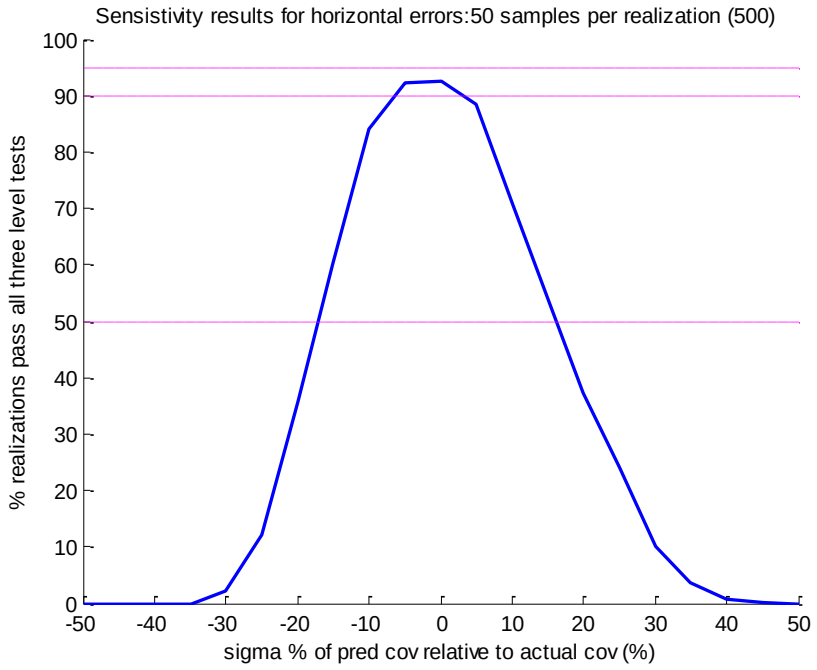


Figure 5.4.2-4: Probability of passing all three normalized error tests; high fidelity, 50 samples

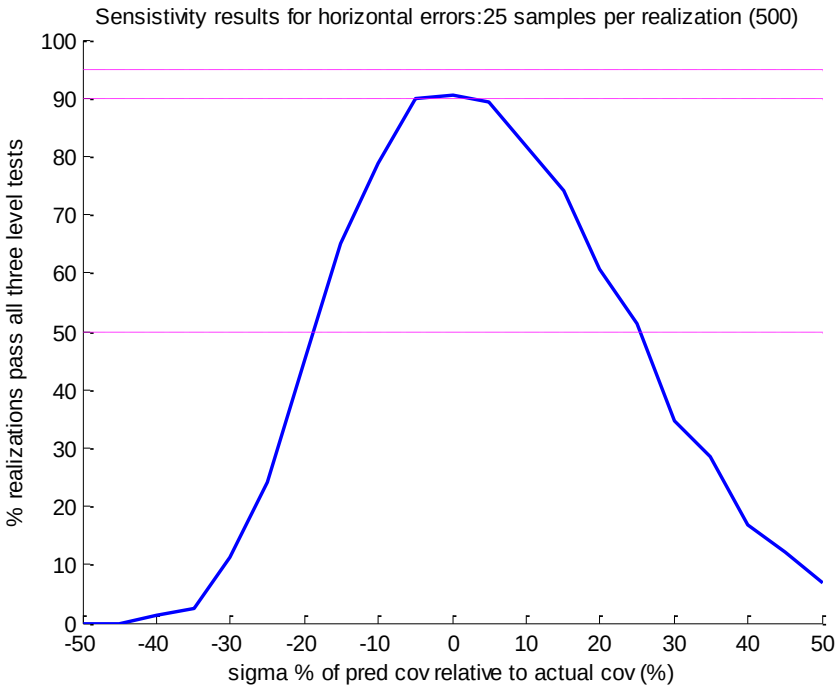


Figure 5.4.2-5: Probability of passing all three normalized error tests; high fidelity, 25 samples

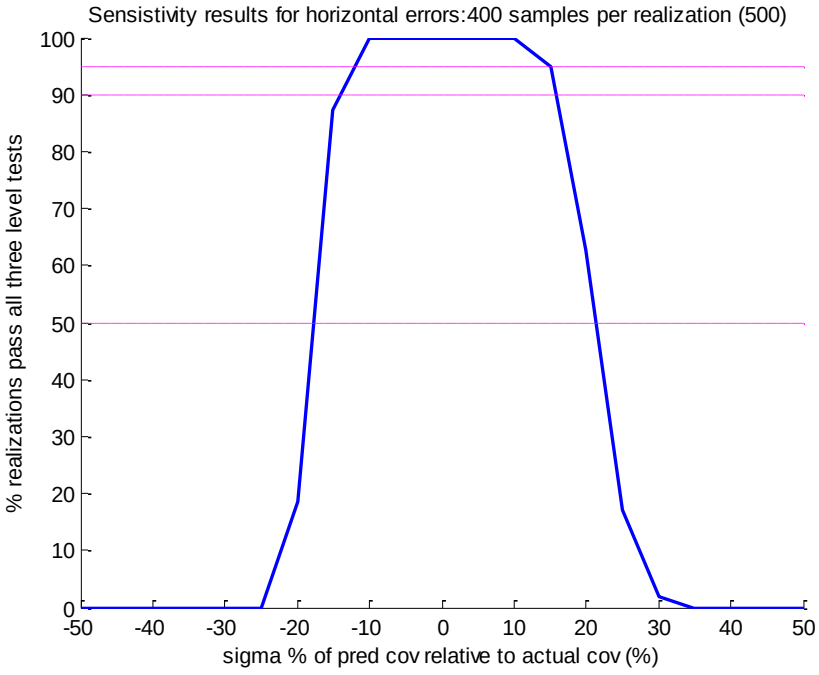


Figure 5.4.2-6: Probability of passing all three normalized error tests; medium fidelity, 400 samples

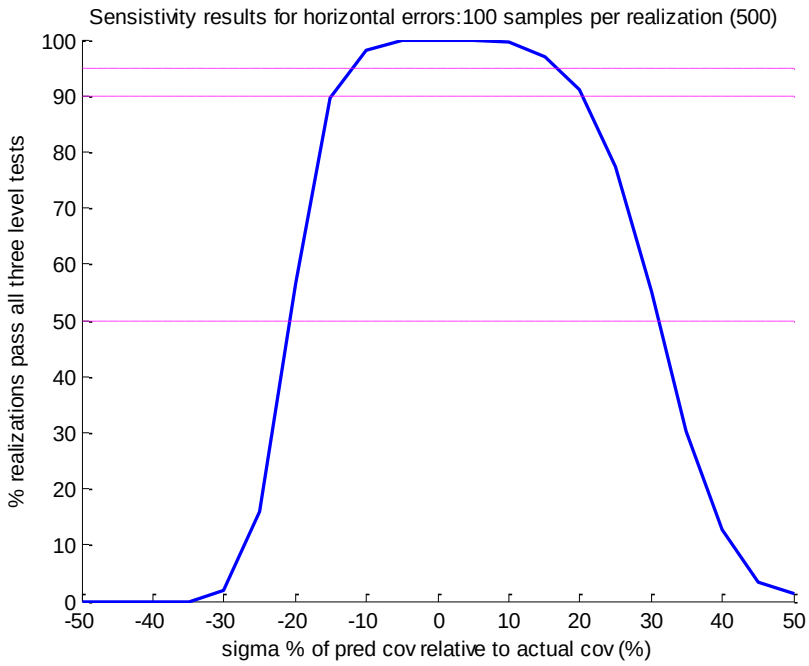


Figure 5.4.2-7: Probability of passing all three normalized error tests; medium fidelity, 100 samples

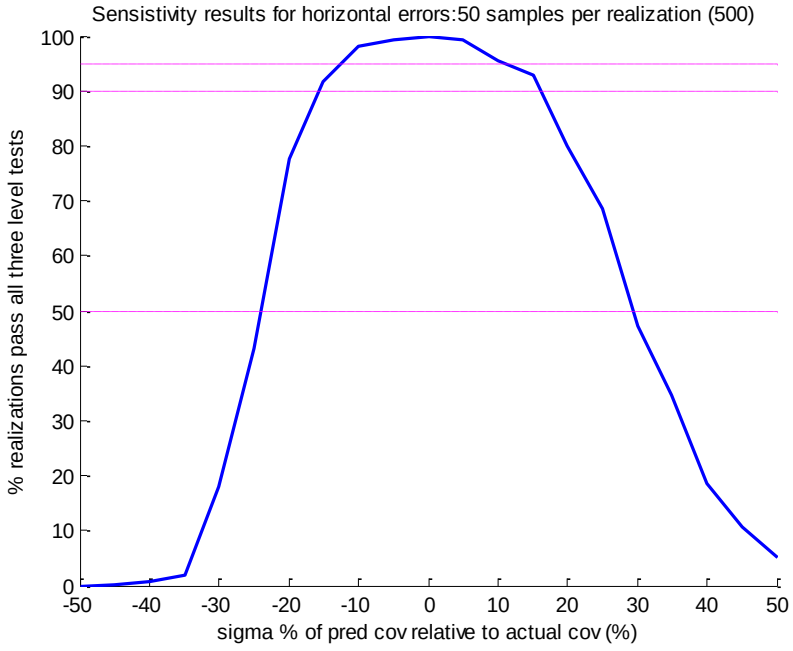


Figure 5.4.2-8: Probability of passing all three normalized error tests; medium fidelity, 50 samples

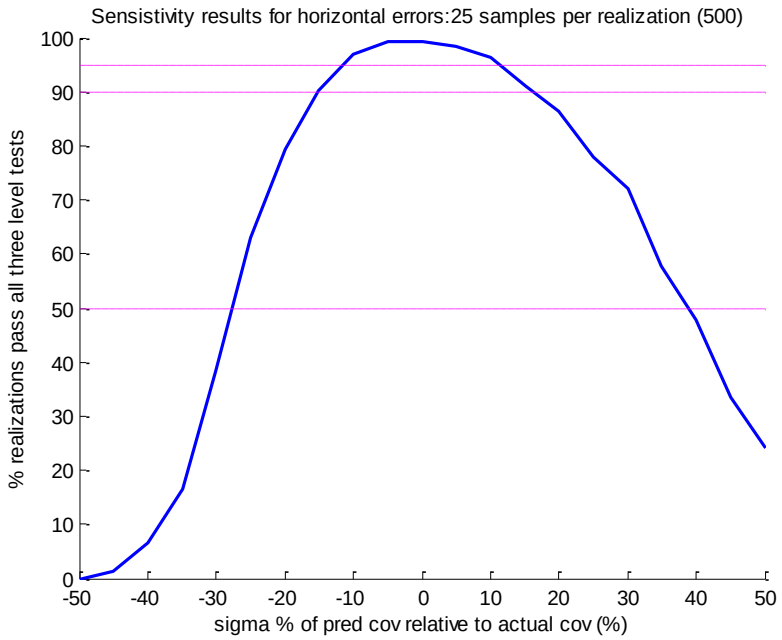


Figure 5.4.2-9: Probability of passing all three normalized error tests; medium fidelity, 25 samples

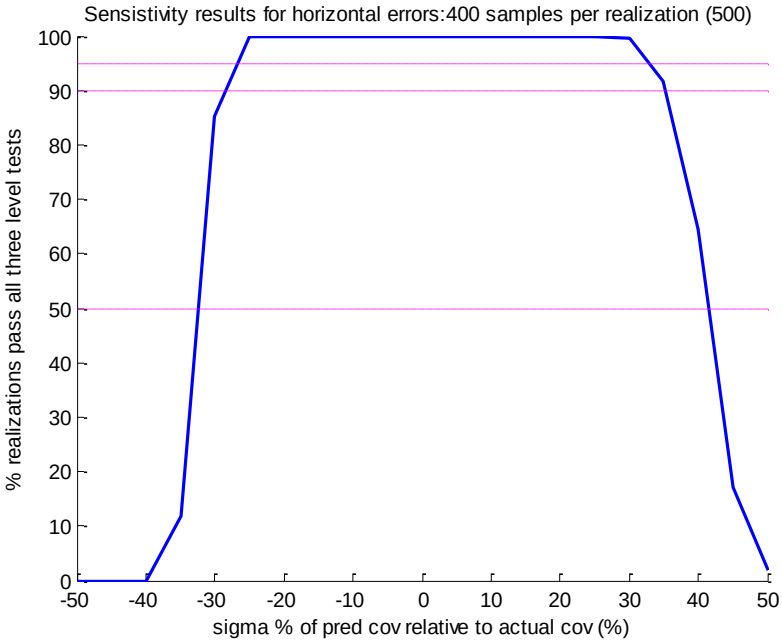


Figure 5.4.2-10: Probability of passing all three normalized error tests; low fidelity, 400 samples

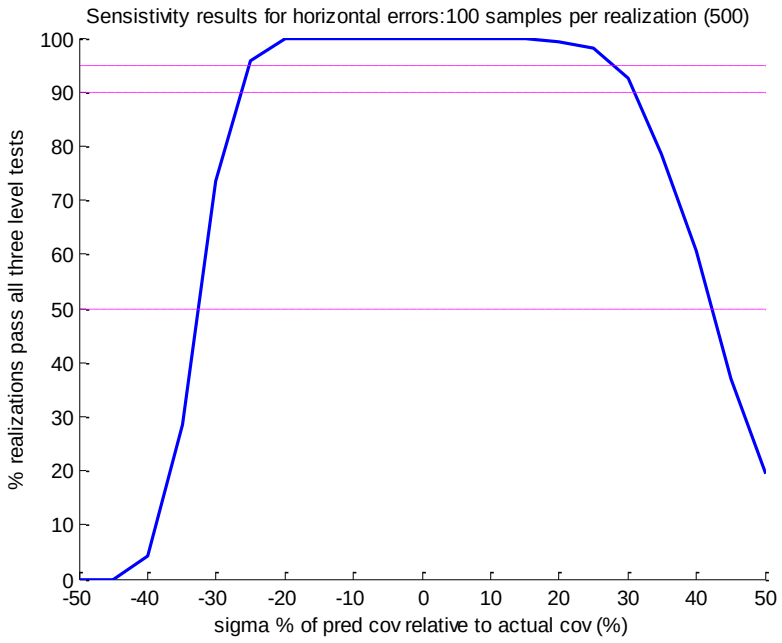


Figure 5.4.2-11: Probability of passing all three normalized error tests; low fidelity, 100 samples

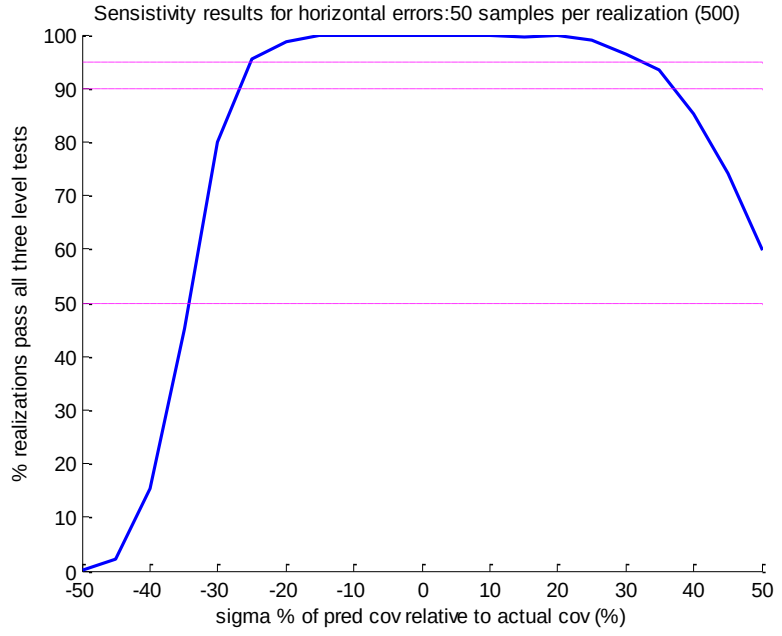


Figure 5.4.2-12: Probability of passing all three normalized error tests; low fidelity, 50 samples

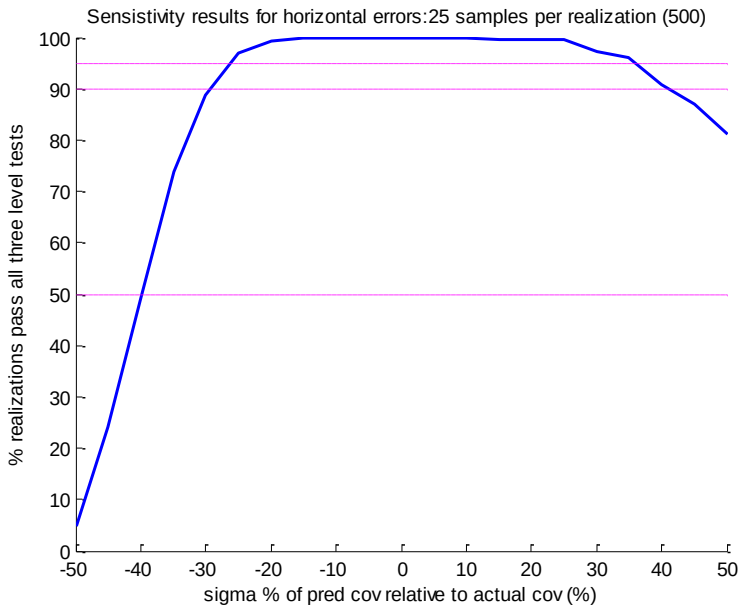


Figure 5.4.2-13: Probability of passing all three normalized error tests; low fidelity, 25 samples

Results of Tables 5.4.2-2 – 5.4.2-4 can also be interpolated, if need be, by both the applicable number of samples and by "in-between" categories of predicted accuracy fidelity. Tables customized to different levels of predicted accuracy fidelity and/or number of samples can also be generated per the techniques described in Appendix D.

5.4.3 Normalized Error Tolerance Values when Scalar Accuracy Metrics are used for Normalization

Normalization of error samples by the corresponding predicted radial (Equation (5.2.1-2)) is preferred over normalization using the corresponding scalar accuracy metric (Equation (5.2.1-5)). However, the latter is not incorrect assuming that the scalar accuracy metric is computed correctly from the corresponding predicted error covariance matrix. This computation (see TGD 2a) automatically takes into account a significant contributing factor: "sqrt_eigen_ratio", the square root of the ratio of the minimum eigenvalue to the maximum eigenvalue contained in the predicted error covariance matrix. For horizontal errors, this corresponds to the ratio of the error ellipse's semi-minor axis to semi-major axis. Figure 5.4.3-1 illustrates this ratio relative to an error ellipse and its corresponding CE counterpart:

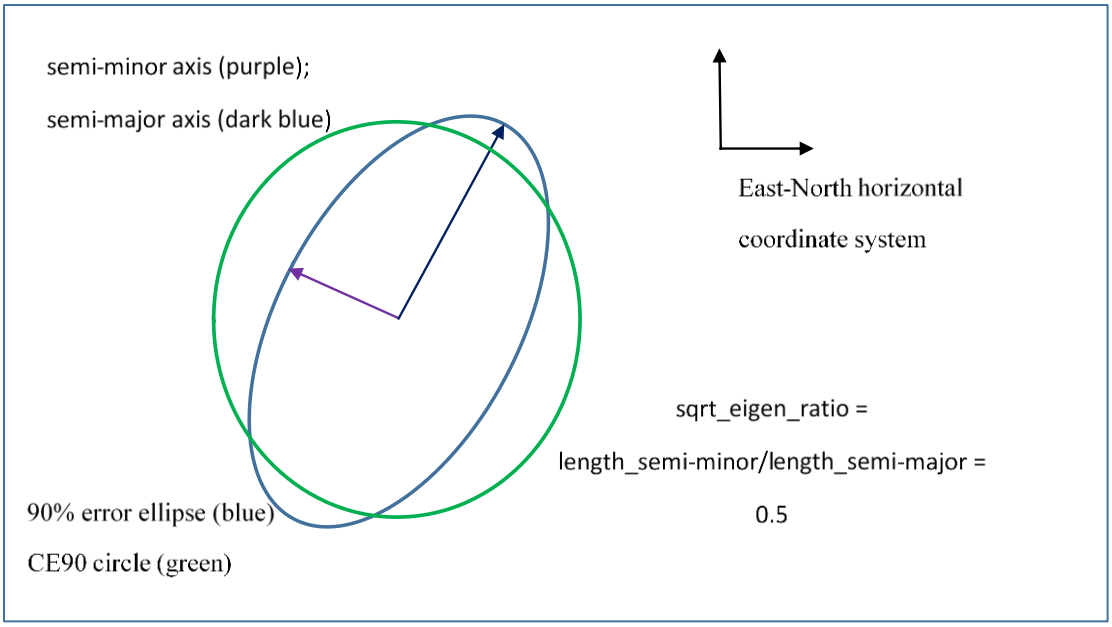


Figure 5.4.3-1: 90% Error Ellipse and Corresponding CE90

The effect of this ratio on the three probability-level tests using scalar accuracy metrics is detailed in Appendix E. What we are interested in are the appropriate normalized error tolerance requirements – do they differ from those corresponding to normalization of errors via predicted radials, assuming the same number of samples and desired predicted accuracy fidelity?

The "good news" is as follows:

- For vertical errors, the corresponding normalized error tolerance requirements of Table 5.4.2-3 are still equally valid.
- For horizontal errors, the results of the three probability-level tests are identical to those using predicted radials when sigma-deviation is 0; however, there are different sensitivities regarding non-zero sigma deviations. If sqrt_eigen_ratio ("ratio") is 0.8 or greater, sensitivities are virtually identical, thus Table 5.4.2-2 can still be used. If ratio is 0.5 or greater, sensitivities are approximately the same, thus Table 5.4.2-2 can still be used as well with some caution (larger "roll-off" or Type II errors, as illustrated in plots below).
- For 3d errors, the conclusions for horizontal errors are approximately applicable as well. However, it is unusual to perform normalization of 3d errors using scalar accuracy metrics (SEXX); hence, this is not be pursued in further detail.

The "bad news" is as follows:

- For ratios smaller than approximately 0.5, custom tables containing normalized error tolerance requirements for normalization of horizontal errors using scalar accuracy metrics are required, and are further parameterized by the value of "ratio". This can be done using the methods presented in Appendix D. However, this becomes complicated and further requires not only knowledge of "ratio" via the assumed availability of the predicted error covariance matrix, but

must assume that this ratio doesn't change significantly over the samples/operational constraints. (Results, however, are invariant to a rotation of the error ellipse or ellipsoid about the ENU axis.)

- The above is another reason why normalization of errors using predicted radials is preferred.

Further note that if only CE90 is available to the validation process (the predicted error covariance matrix is not):

- CE99 and CE50 can be estimated from CE90 using an assumed ratio of 1:
  - $CE99 = \left(\frac{3.035}{2.146}\right) CE90, CE50 = \left(\frac{1.177}{2.146}\right) CE90$
- Of course, per the earlier discussion, results will only be acceptable if it is known *a priori* that the actual ratio is greater or equal to 0.8.

The effects of the above issues and conclusions are illustrated in the following plots for normalized horizontal error assuming 100 i.i.d. error samples, similar to the earlier plots Figure 5.4.2-1 through 5.4.2-13. A blue curve corresponds to the use of predicted radials for normalization (illustrated in the earlier plots), a green line to the use of correctly computed CE99, CE90, and CE50, and a red line to the use of a correctly computed CE90, but a CE99 and a CE50 computed from CE90 assuming a specified ratio value.

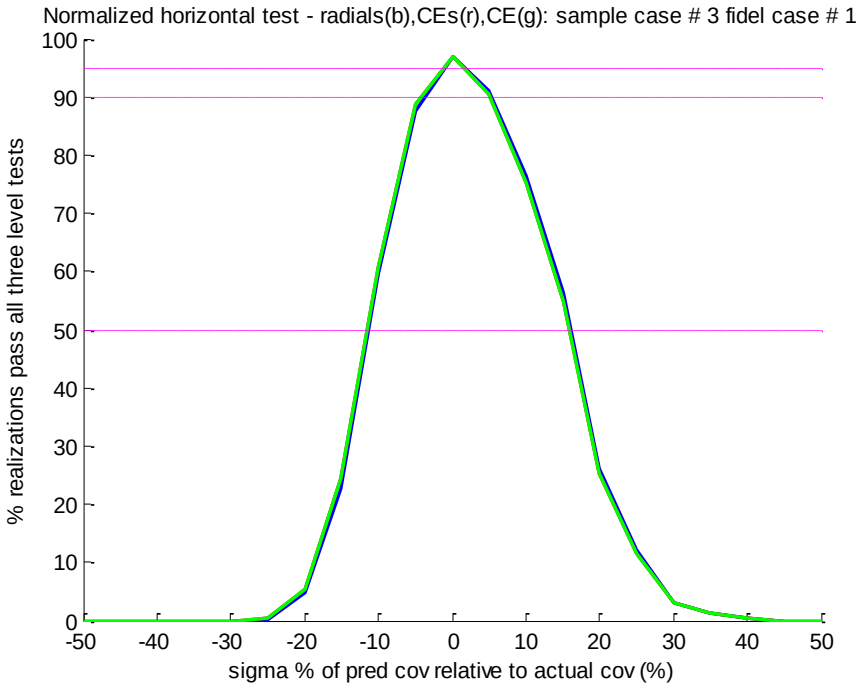


Figure 5.4.3-1: 100 samples and high fidelity; actual ratio = 1 and assumed ratio =1

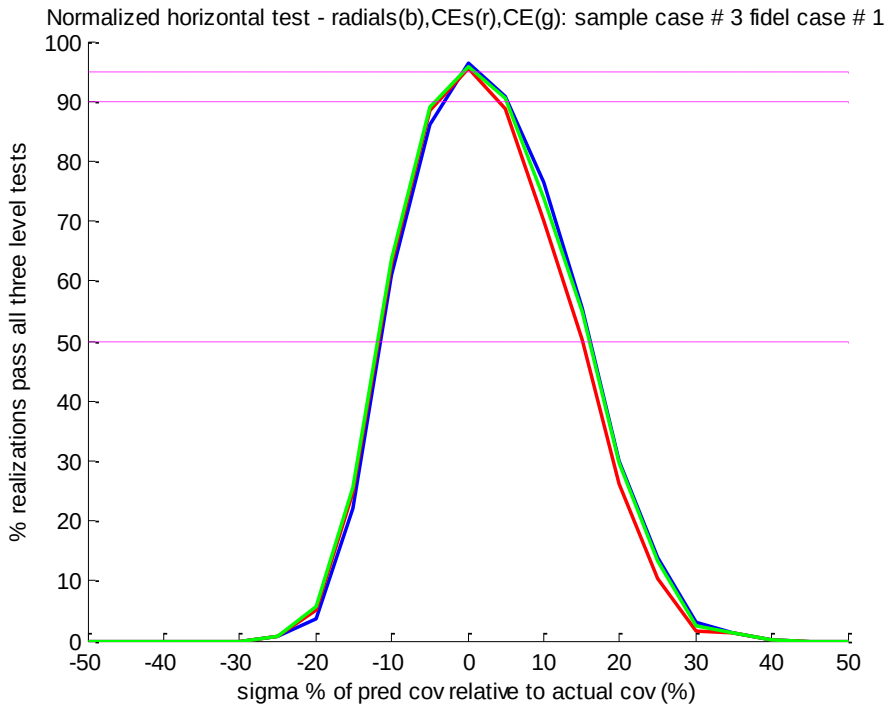


Figure 5.4.3-2: 100 samples and high fidelity; actual ratio = 0.8 and assumed ratio =1

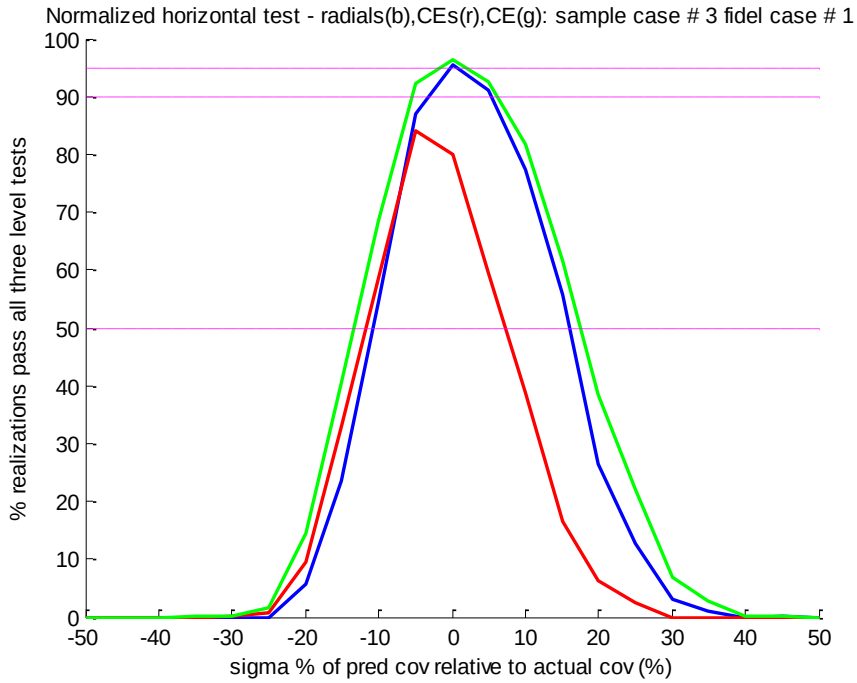


Figure 5.4.3-3: 100 samples and high fidelity; actual ratio = 0.5 and assumed ratio =1

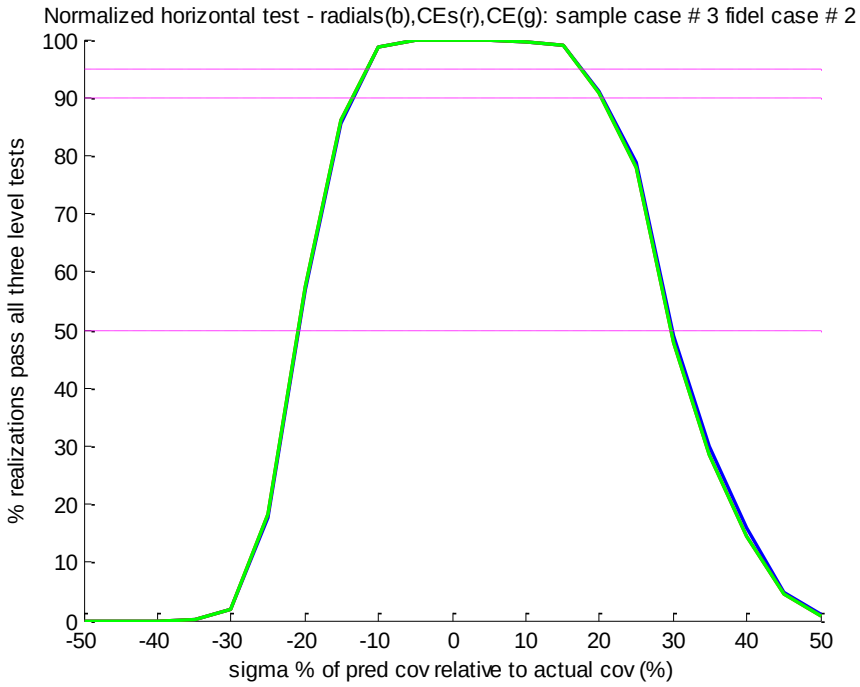


Figure 5.4.3-4: 100 samples and medium fidelity; actual ratio = 1 and assumed ratio =1

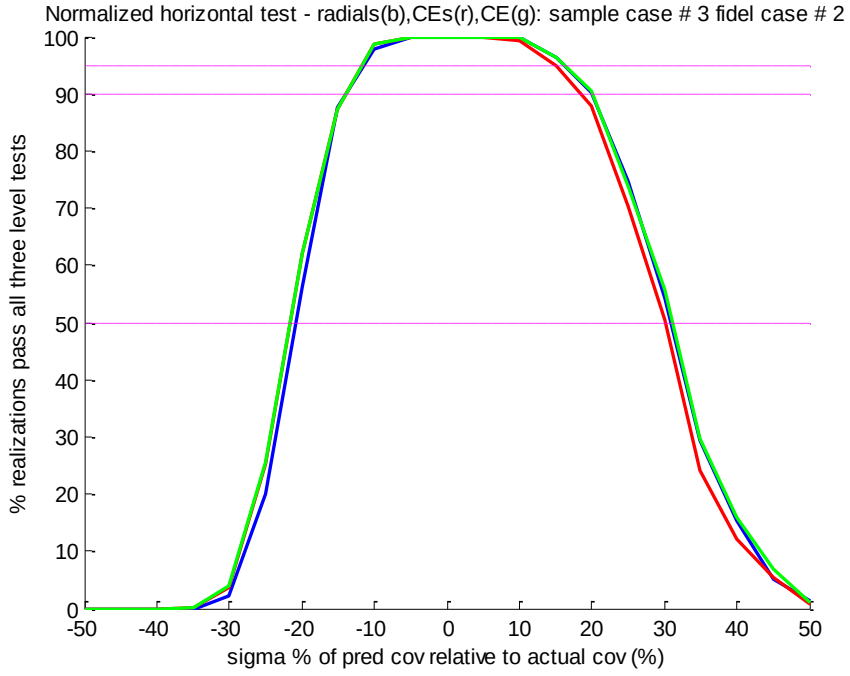


Figure 5.4.3-5: 100 samples and medium fidelity; actual ratio = 0.8 and assumed ratio =1

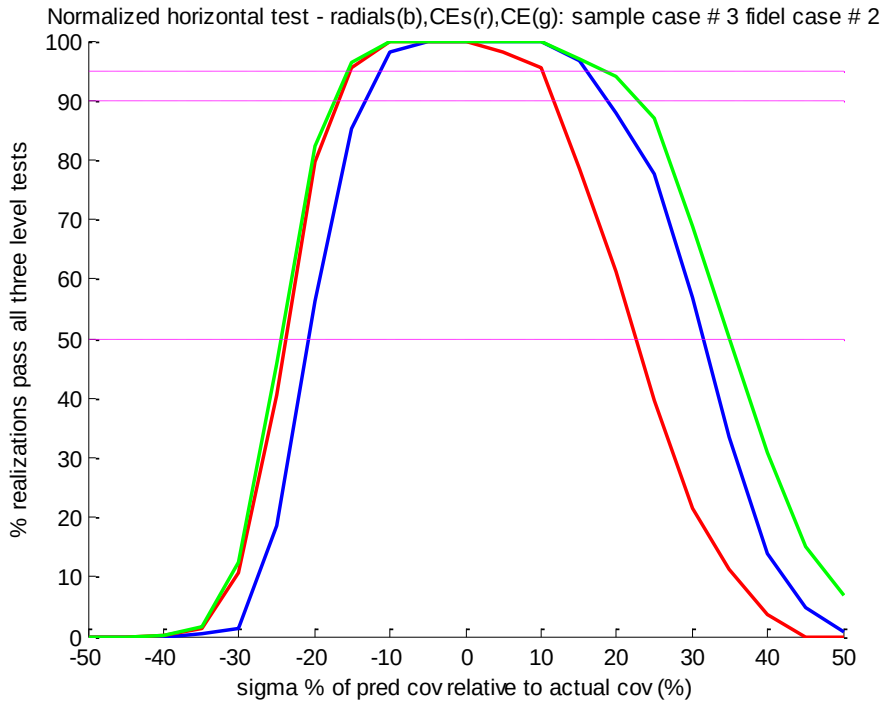


Figure 5.4.3-6: 100 samples and medium fidelity; actual ratio = 0.5 and assumed ratio =1

Thus, the applicable table for normalized error tolerance values for horizontal errors normalized using scalar accuracy metrics (CEXX) corresponds to Table 5.4.2-2 applicable to predicted radials. However, as discussed earlier, this also assumes that ratio (sqrt_eigen_ratio) is greater than or equal to approximately 0.8, and possibly 0.5 with some extra "roll off". This also assumes that CE99, CE90, and CE50 are computed correctly. If CE99 and CE50 must be derived from CE90 from scale factors based on an assumed ratio of 1, the (actual) ratio must be approximately 0.8 or greater for reasonable results. (Plots 5.4.3-1 through 5.4.3-6 above were generated using "pseudo-code" documented in Appendix E.)

In summary, the use of predicted scalar metrics is not part of the baseline approach for the validation of predicted accuracy. However, it can be used if need be. For example, if the predicted error covariance matrix is unavailable to the validation process but corresponding predicted scalar accuracy metrics are available instead (assumed generated correctly from the predicted error covariance matrix by some other application/process). This is also predicated on *a priori* knowledge that ratio (sqrt_eigen_ratio) is not too small as discussed earlier.

5.4.4 Recommended Future Research

This section discusses recommended future research corresponding to the specification and validation of predicted accuracy:

It is recommended that both: (1) the definitions of low, medium, and high predicted accuracy fidelity as a function of sigma deviation range, and (2) the corresponding normalized error tolerance values (e.g.,

$\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}$ $\}$) be adjusted or "fine-tuned" based on future research. The former based on additional input regarding applicable "classes" of NSG geolocation system mission requirements regarding predicted accuracy fidelity, and the latter based on both formal and informal predicted accuracy validation results using real data, as well as extension of the simulations discussed in Appendix D..

In addition, it is recommend that future research also address the assumption that underlying error-components are mean-zero multi-variate Gaussian. This assumption directly affects the computation of the normalized errors, Equations 5.2.1-2 through 5.2.1-4, and Equation 5.2.1-3 in particular, as well as the Monte-Carlo simulation results used to generate the normalized error tolerance values, e.g. $\{YY_{h\_99\_spec}$, $YY_{h\_90\_spec}$, $YY_{h\_50\_spec}$ $\}$. In particular, how robust are predicted accuracy validation results regarding this assumption, and possible modifications of the validation algorithm/parameters if given *a priori* information of a different (possibly empirical) distribution?

It is hypothesized that multi-variate Gaussian distributions with non-zero means of reasonable magnitude are not a problem – the current approach to the specification and validation of predicted accuracy should be reasonably robust if this is the case. This follows from considering the predicted error covariance as an approximate true error covariance matrix + $\overline{\epsilon X}\,\overline{\epsilon X}^T$, where $\overline{\epsilon X}$ is the true mean-value of geolocation error. The use of this predicited error covariance matrix along with an assumed mean-value of zero fits well within the sigma deviation paradigm. Preliminary verification has been performed by simulation.

It is hypothesized that the suitability of the multi-variate Gaussian distribution assumption is dependent on the absence of more extreme sensor-to-geolocation geometries and sensor orientations where non-trivial non-linear effects may occur. In addition, the presence of higher probabilities for more extreme values of error are of concern, as well as the applicability of significantly different probability distributions, such as a "Mixture of Gaussians" distribution: $pdf(\epsilon X) = \sum_{i=1}^{N} w_i pdf_G(\epsilon X; \mu_i, \Sigma_i)$, where $\sum_{i=1}^{N} w_i = 1$. The "Mixture of Gaussians" is still a valid multi-variate probability distribution but can have multiple modes, with each mode being a Gaussian with probability density function $pdf_G(\epsilon X; \mu_i, \Sigma_i)$. The whole pdf itself is not Gaussian even though the individual components are.

Despite these potential weaknesses, it is also hypothesized that the major features of the current approach to the specification and validation of predicted accuracy can be successfully retained as they were designed with robustness in mind: (1) the use of three normalized error probability test levels and corresponding one-sided tests (two below, one above), and (2) the use of predicted accuracy fidelity categories with corresponding sigma deviation scaling of error covariance matrices. This does not rule out that some changes to the current approach and related parameter values will be needed.

It is also noted that non-Gaussian probability distributions of error may still be represented (approximated) by a MIG-type estimator's *a posteriori* covariance matrix (aka predicted error covariance matrix) for practicality. Furthermore, the most general MIG-type estimator is a Best Linear Unbiased Estimator (minimum variance estimator) – it need not assume a Gaussian distribution of errors.

5.5 Relationships between Predicted Accuracy Validation Tests, Plotting, and the Chi-Square Probability Distribution

The three probability-level normalized error tests (Equation (5.2.2-1)) for an error of interest (vertical, horizontal, or 3d) are convenient, practical, and plot-friendly tests for the validation of predicted accuracy. They are one-sided tests that work in concert together to ensure reasonable normalized errors, i.e., reliable predicted accuracies.

Figure 5.5-1 presents a conceptual graphical depiction of the tests assuming 3d errors. (See Figure 5.2.3.2-4 for an actual depiction and Figures 5.2.3.2-1 through 5.2.3.2-3 for single-test counterparts.) The 99% probability normalized error test is equivalent to the percentage of radial error samples below the blue line that has the largest slope in Figure 5.5-1. This percentage must be greater than or equal to the specified tolerance $YY_{r\_99\_spec}$ which has a typical value of 95%. This test ensures that few if any normalized error samples are excessively large, such as those corresponding to the red circles of 3d radial error samples in Figure 5.5-1. The 90% probability normalized error test is equivalent to the percentage of radial error samples below the blue line that has the slope of 1. This percentage must be greater than or equal to the specified tolerance $YY_{r\_90\_spec}$ which has a typical value of 83%. This test ensures that a reasonable number of radial error samples are smaller than their predicted 90% probable magnitudes, and are represented symbolically by a significant subset of the blue circles in Figure 5.5-1. The 50% probability normalized error test is equivalent to the percentage of radial error samples above (not below) the blue line with the smallest slope. This percentage must be greater than or equal to the specified tolerance $YY_{r\_50\_spec}$ which has a typical value of 39%. This test ensures that predicted radials are not excessively large, such as the dark red circles corresponding to 3d radial error samples in Figure 5.5-1. This test "balances" the two other tests.

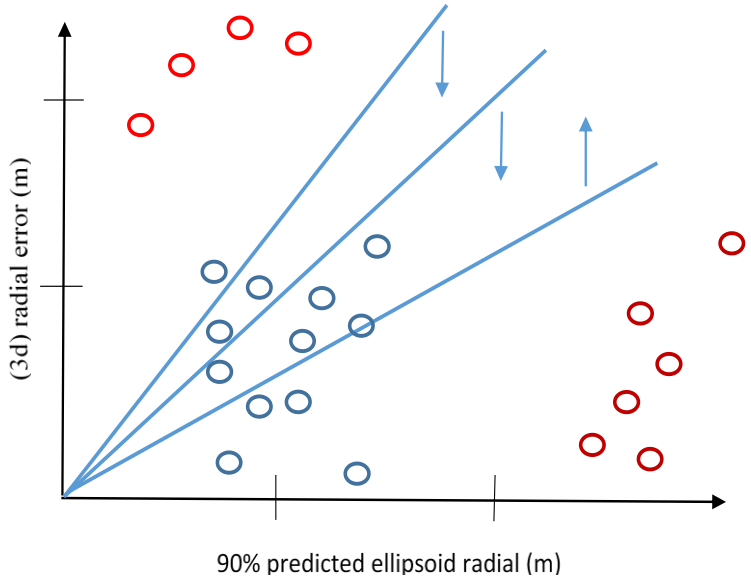


Figure 5.5-1: Conceptual graphic of the three probability-level normalized error tests for validation of predicted accuracy

A similar concept is applicable to errors normalized by scalar accuracy metrics (e.g., SEXX for 3d errors or CEXX for horizontal errors); however, as discussed earlier, a combined graphic corresponding to all three probability-levels is not possible, and three separate graphics must be generated.

Also, keep in mind that the specified values for tolerances $\{YY_{r\_99\_spec}$, $YY_{r\_90\_spec}$, $YY_{r\_50\_spec}$ $\}$ are selected based on the expected number of samples available to validation as well as the desired level of predicted accuracy fidelity as discussed earlier in Section 5.4.2. That is, they are intended to be practical values, not simply selected based on theory corresponding to an unlimited number of samples and assumed perfect (probabilistic-based) predicted accuracy fidelity. Another interesting point is that the normalized (radial) errors squared, assuming a multi-variate, mean-zero, Gaussian probability distribution for the underlying error vector $\epsilon X = [\epsilon x \quad \epsilon y \quad \epsilon z]^T$, corresponds to a Chi-square probability distribution with n degrees of freedom (dof), where n is the vector dimension of ϵX, e.g., $n = 2$ if horizontal normalized errors are of interest. This is discussed in [7], with Figure 5.5-2 below an excerpt. Reference [2] also discusses the Chi-square distribution in general. The figure actually presents the Chi (not Chi-squared) probability distribution for normalized errors since it is a function of normalized error magnitude (distance) d (meters) instead of normalized error squared-magnitude d^2 (d^2 not directly related to d_{XX}^2 presented earlier in this document).

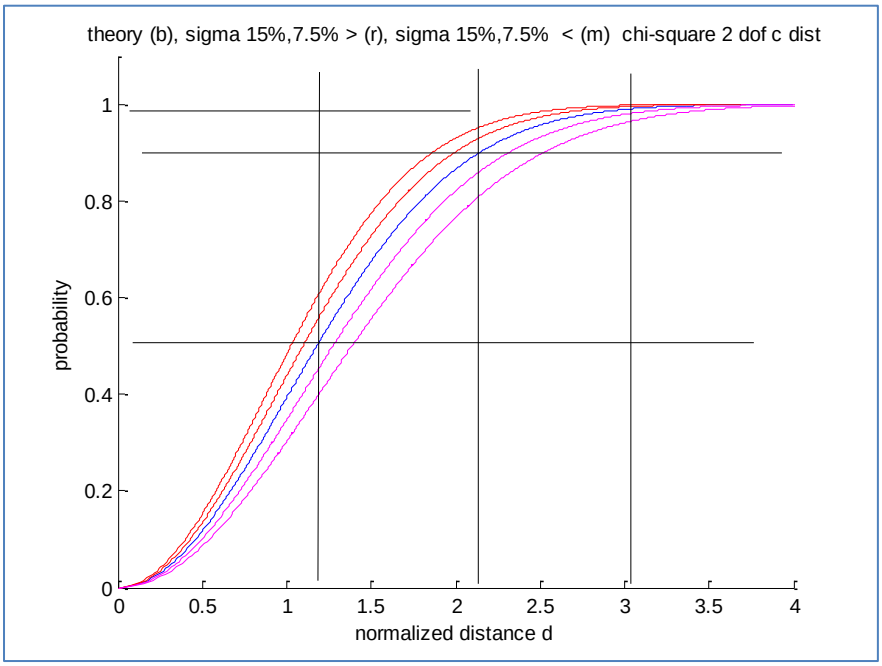


Figure 5.5-2: Chi probability distribution function (2 degrees of freedom)

The blue (middle) curve in the above figure corresponds to the theoretical probability distribution assuming an unlimited number of samples and perfect predicted accuracy fidelity. The surrounding curves correspond to +/- 7.5% (middle curves) and +/- 15% (outer curves) sigma deviations. Their intersection with the horizontal 99%, 90%, and 50% probability levels and their perpendicular lines correspond to test tolerances at the three probability-levels for practicality – see [7] for more details.

Also, although not shown, sample-based Chi probability distributions are also easily generated by ordering i.i.d. samples of the corresponding normalized error. Of course, they are much less smooth than their theoretical counterparts as a function of the number of samples – the more the samples the smoother the function.

(As an interesting aside and as discussed in TGD 2b, the radial errors themselves, $\epsilon v, \epsilon h, and\ \epsilon r$, have folded-Normal, Rayleigh, and Maxwell probability distributions respectively, again assuming a multi-variate, mean-zero, Gaussian distribution for the underlying error components contained in vector ϵX, including the additional constraint that the corresponding *a priori* error covariance matrix is a diagonal matrix with equal-valued diagonal components or component variances.)

5.6 Processing Multiple Error Samples from the Same Set of Sensor Data: Correlated Error Samples

This document has assumed the processing of i.i.d. error samples for both the validation of accuracy and the validation of predicted accuracy. However, it is not uncommon in practice that for each i.i.d. error sample, a sub-collection of correlated error samples is also available. For example, there could be 100 i.i.d. error samples, each associated with another 5 error samples that are positively correlated with the original error sample, and of course, with themselves. More specifically, for example, a geolocation system with extraction based on the use of one (mono) or a stereo pair of images, a total of 6 geolocations could be extracted per image or stereo pair of images (on average) with corresponding ground truth available in order to compute error samples. These 6 geolocations are correlated since they are based on the same data (images and image support data). (They are also assumed uncorrelated with all other groups of 6 error samples since the corresponding imagery was assumed imaged over different satellite passes.) What to do, if anything, with the extra 500 error samples that are available in this example?

There are three basic approaches, each assumed, as a minimum, to use the additional error samples in support of a reasonableness check for the value of the selected original i.i.d. error sample; if deemed an outlier, replaced by one of the correlated error samples. In the above example, an outlier typically occurs due to either mis-measurement of the target's location in the image(s), possibly due to mis-identification of the ground truth point's location in the image(s), or possibly due to a faulty ground truth point location (or coordinate values(s)) itself.

Prior to describing these three approaches, Subsection 5.6.1 first illustrates the importance of using i.i.d. error samples in general for the validation of accuracy per the baseline method documented in Section 4.1/5.1. I.i.d. error samples are also important for the validation of predicted accuracy as well, but are not illustrated explicitly in this subsection.

Subsections 5.6.2 – 5.6.4 then go on to provide a description of each of the three basic approaches to deal with sub-collections of correlated samples. The first approach, "i.i.d. error samples only", is the preferred approach and considered the baseline for sub-collections of correlated samples. These

subsections exclude a description of possible reasonableness checks, as many different approaches are both applicable and readily available. For example, and for a reasonable number of sub-collections, such a check could simply entail plotting all of the error samples in a correlated sub-collection and look for those that are too big and inconsistent with the others.

Note that various pseudo-code in support of Subsections 5.6.1-5.6.4 are documented in Appendix G.

5.6.1 The importance of i.i.d. error samples in general

This subsection illustrates the importance of using i.i.d. error samples in general, and specifically, the effects of both i.i.d. samples and non-i.i.d. (correlated) samples on validation of accuracy using the baseline validation procedure of Subsection 4.1/5.1.

60 samples of correlated horizontal errors were simulated (including zero correlation or i.i.d. samples), with both the best estimate and the lub (90% confidence) of the 90th percentile of horizontal radial error computed per the baseline algorithm of Subsections 4.1/5.1. 60 samples were computed for each of 1000 independent realizations. There were no correlated sub-collections, or alternatively and equivalently, there was only one correlated sub-collection containing 60 correlated samples for a given realization. Underlying x and y error components were simulated based on a multi-variate Gaussian distribution and consistent with a true value of the 90th percentile of horizontal radial error ϵh_{90} equal to 2.14 m. The actual generation of 60 2d correlated error samples was based on the method presented in TGD 2e (Monte Carlo simulation) and summarized as follows for realization # m:

$$X_m = C_X^{1/2} r, \qquad 5.6.1\text{-}1$$

where X_m is a $120x1$ vector of 60 2d correlated error samples, r is a $120x1$ vector of independent realizations of a Gaussian random variable with mean-value of zero and variance of 1, and $C_X^{1/2}$ is the principal matrix square-root of the $120x120$ joint covariance matrix C_X, with common $2x2$ diagonal blocks equal to cov consistent with the (simulation data base parameter) value of ϵh_{90}, and with common $2x2$ off-diagonal blocks equal to $crosscov = \rho \cdot cov$, where ρ is the specified common correlation (coefficient) between each pair of 2d error samples and which multiplies each element of the 2×2 matrix cov:

$$C_X = \begin{bmatrix} cov & \rho \cdot cov & \dots & \rho \cdot cov \\ \rho \cdot cov & cov & \dots & \rho \cdot cov \\ \dots & \dots & \dots & \dots \\ \rho \cdot cov & \rho \cdot cov & \dots & cov \end{bmatrix}. \qquad 5.6.1\text{-}2$$

Histogram results were then generated and plotted with the value of ϵh_{90} the vertical red line, and approximately 90% of lub's larger than the magenta line (to the right of this line). The magenta line must be (slightly) greater than the red line for successful validation at the 90% probability level, i.e., slightly more than 90% of the lub's should be greater than the red line (ϵh_{90}).

The following presents results for three separate sub-cases, where common correlation between each of the 60 samples was equal to either 0 (i.i.d. samples), 50, or 90% correlation ($\rho = 0, 0.50, 0.90$). Each

sub-case first presents a plot of the histogram containing 1000 best estimates of ϵh_{90}, followed by a plot of the histogram containing 1000 different lub's for ϵh_{90}.

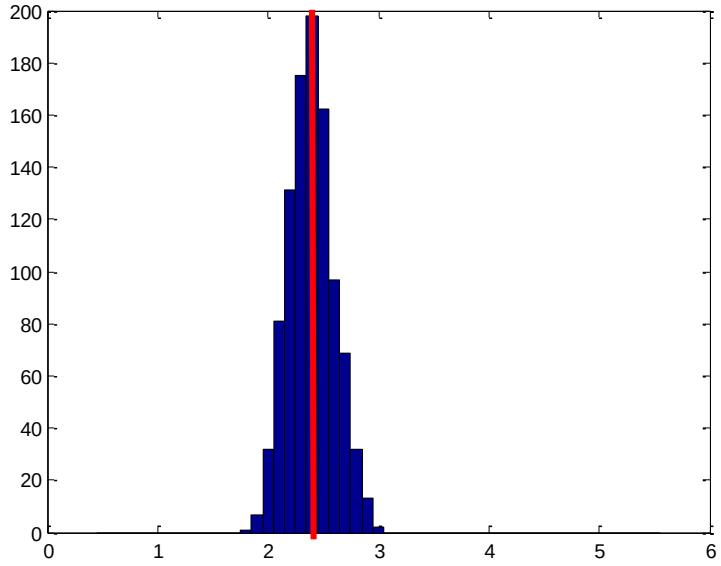


Figure 5.6.2-1: Histogram of best estimate of eh_90: 0 correlation between 60 (i.i.d.) samples per each of 1000 independent realizations; red line corresponds to true value of eh_90; results successful as expected – best estimates close to true value

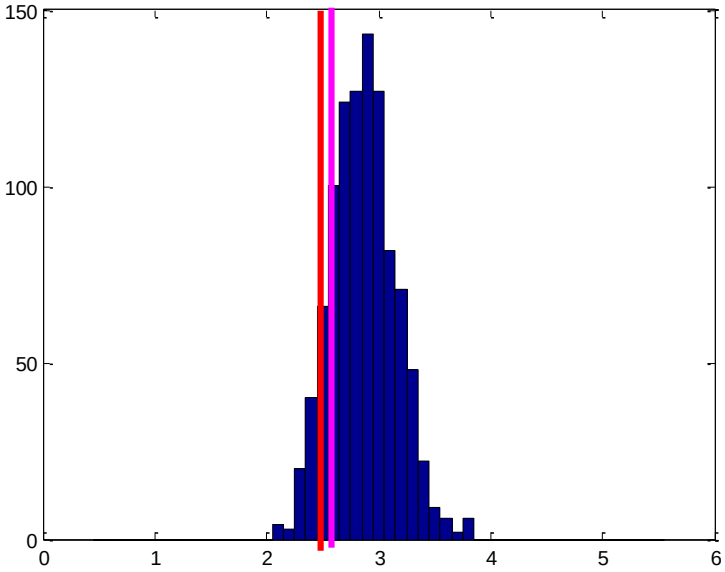


Figure 5.6.2-2: Histogram of lub of eh_90: 0 correlation between 60 (i.i.d.) samples per each of 1000 independent realizations; red line corresponds to true value of eh_90; 90% of lubs greater than magenta line; results successful as expected as approximately 92% of lubs slightly greater than red line

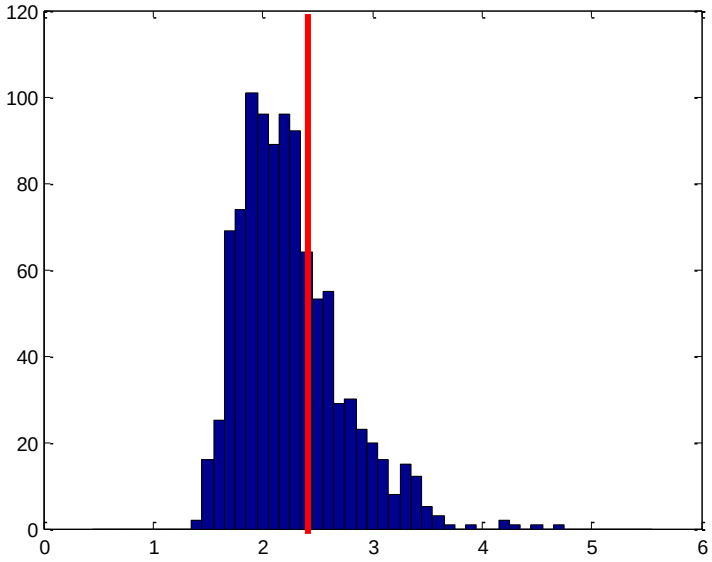


Figure 5.6.2-3: Histogram of best estimate of eh_90: 50% correlation between 60 samples per each of 1000 independent realizations; red line corresponds to true value of eh_90; results starting to degrade – too large of spread of best estimates around true value

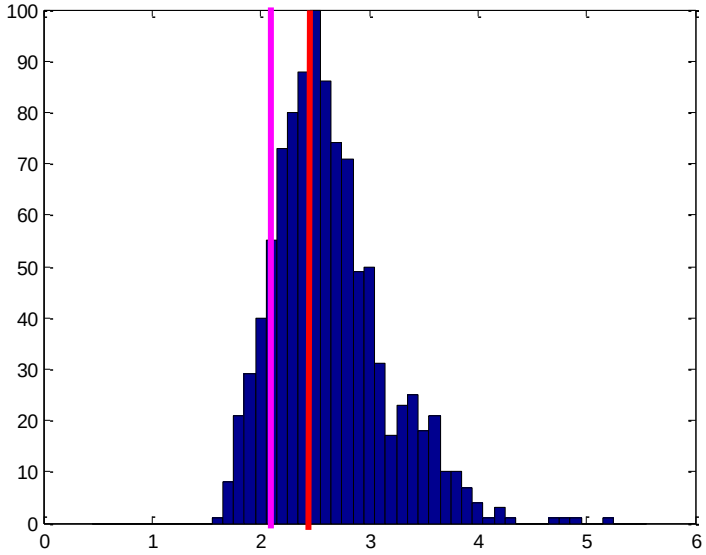


Figure 5.6.2-4: Histogram of lub of eh_90: 50% correlation between 60 samples per each of 1000 independent realizations; red line corresponds to true value of eh_90; 90% of lubs greater than magenta line; results poor as approximately only 60% of lubs greater than red line

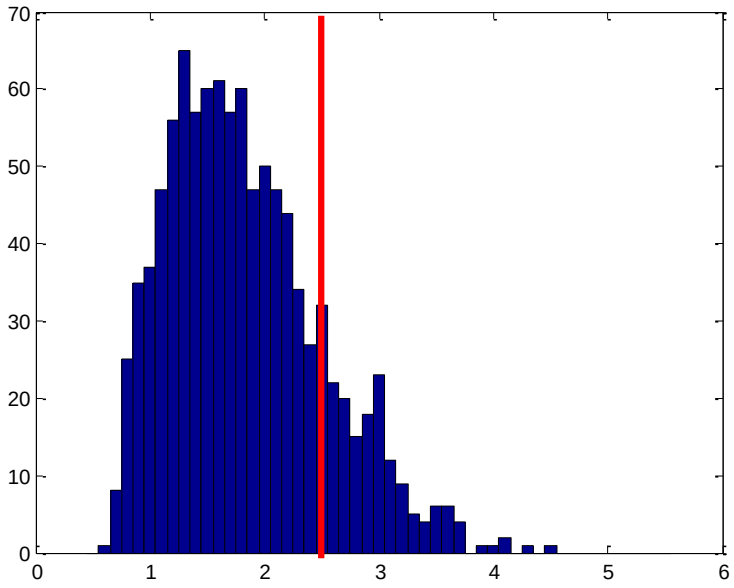


Figure 5.6.2-5: Histogram of best estimate of eh_90: 90% correlation between 60 samples per each of 1000 independent realizations; red line corresponds to true value of eh_90; results very poor – very wide spread of best estimates around true value

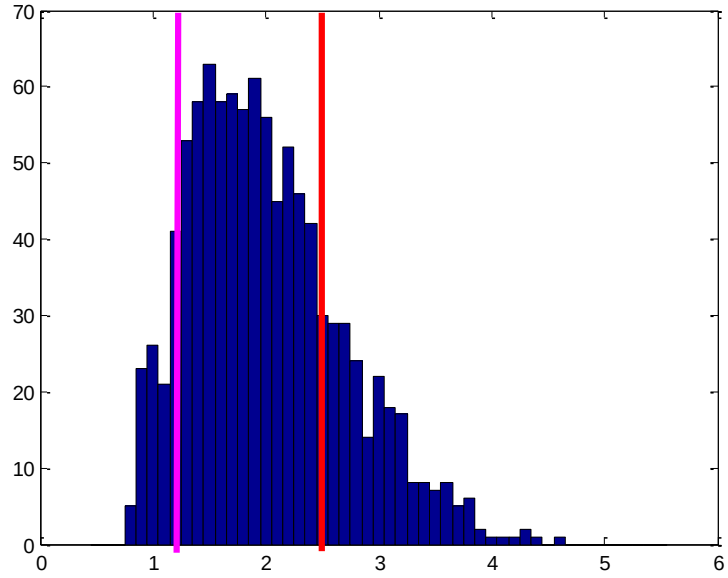


Figure 5.6.2-6: Histogram of lub of eh_90: 90% correlation between 60 samples per each of 1000 independent realizations; red line corresponds to true value of eh_90; 90% of lubs greater than magenta line; results very poor as approximately only 25% of lubs greater than red line

The following results correspond to the same general experiment as above except that there are 10 correlated sub-collections, each with 6 correlated samples, and each sub-collection independent (uncorrelated) with the others. Results are first presented for 50% correlation within a correlated sub-collection followed by 90% correlation within a correlated sub-collection. In general, results are

improved. For 50% correlation within a correlated sub-collection, results are close to the desired results corresponding to i.i.d. samples (Figure 5.6.2-1 and 5.6.2-2). For 90% correlation within a correlated sub-collection, results are still unacceptable (poor) but better than the earlier results (Figures 5.26.2-5 and 5.6.2-6) which correspond to all 60 samples correlated 90%.

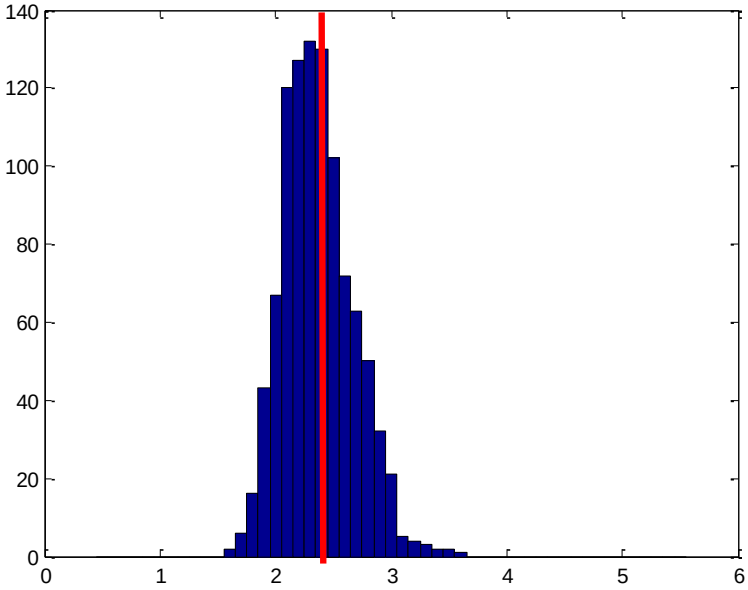


Figure 5.6.2-7: Histogram of best estimate of eh_90: 50% correlation between 6 samples per correlated sub-collection for each of 1000 independent realizations; red line corresponds to true value of eh_90; results acceptable

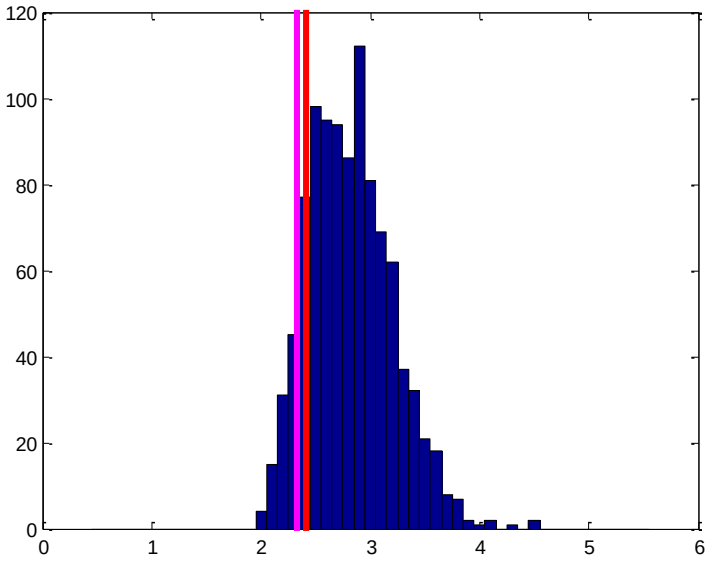


Figure 5.6.2-8: Histogram of lub of eh_90: 50% correlation between 6 samples per each of 10 correlated sub-collections for 1000 independent realizations; red line corresponds to true value of eh_90; 90% of lubs greater than magenta line; results borderline acceptable as approximately 86 % of lubs greater than red line

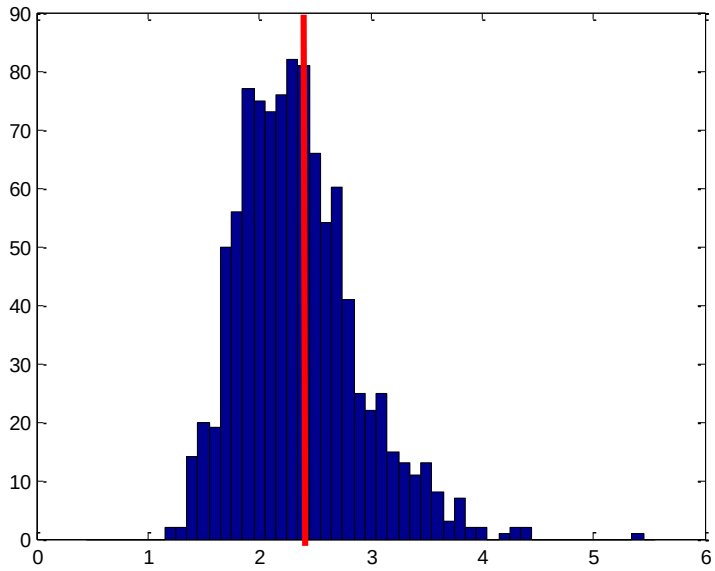


Figure 5.6.2-9: Histogram of best estimate of eh_90: 90% correlation between 6 samples per correlated sub-collection for each of 1000 independent realizations; red line corresponds to true value of eh_90; results poor

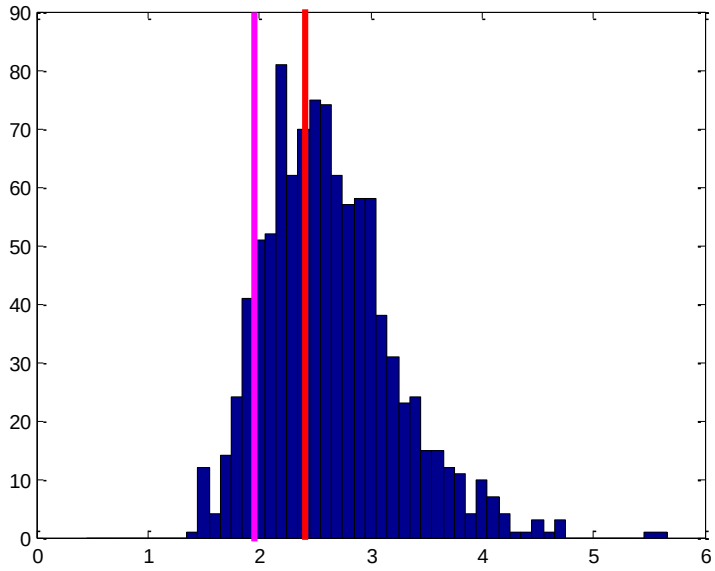


Figure 5.6.2-10: Histogram of lub of eh_90: 90% correlation between 6 samples per each of 10 correlated sub-collections for 1000 independent realizations; red line corresponds to true value of eh_90; 90% of lubs greater than magenta line; results poor as approximately only 60 % of lubs greater than red line

In summary, i.i.d. or nearly i.i.d. error samples are required for proper validation of accuracy. The above figures provided quantitative results for varying levels of correlation based on the baseline validation procedure of Sections 4.1/5.1.

For a given level of correlation (50% or 90%), the use of correlated sub-groups did improve the poor to very poor validation results as they decreased overall "average" correlation. However, they were still unacceptable for 90% correlation within a sub-collection. (Relative improvements may differ with more sub-collections.) And as a reminder, the baseline validation procedure was used in all the above experiments: i.i.d. error samples were assumed and no *a priori* information was utilized regarding error samples grouped into correlated sub-collections.

The following now goes on to address three different approaches for alternative (non-baseline) processing when correlated sub-collections are known to be available, each sub-collection containing correlated error samples. Validation of both accuracy and predicted accuracy requirements are addressed.

5.6.2 i.i.d. Error Samples Only Approach

This approach does not use the extra error samples available in each correlated sub-collection. It performs normal (baseline) validation processing using one i.i.d. error sample from each and all correlated sub-collections. Assuming a reasonable number of corresponding i.i.d. error samples, this is certainly a reasonable approach and theoretically valid with no additional assumptions and approximations necessary for both accuracy and predicted accuracy validation. However, it is recommended that each i.i.d. error sample is either selected randomly from its correlated sub-collection of error samples, or by a deterministic algorithm designed to ensure that the entire "collection solution-space" is sampled over the i.i.d. error samples as much as possible. For example, for a geolocation system based on stereo-pairs of same pass images, i.i.d. samples are selected within different quadrants (or finer subdivision) of the ground footprints of the various independent stereo pairs.

No further examples or details are necessary for this approach, as essentially the entire document up to this subsection has been based on this approach. It is the recommended approach assuming that both the reasonableness checks discussed above are performed and that a reasonable number of i.i.d. samples (sub-collections) are available – at least 40 and preferably 100 or more.

However, there is one caveat – the i.i.d. Error Samples Only approach is applicable to the validation of (absolute) accuracy and predicted accuracy. It is not applicable to the validation of relative accuracy or predicted relative accuracy, as the error samples in a correlated sub-collection must be retained for corresponding processing per Section 5.3.

5.6.3 All Error Samples Approach

This approach uses all available error samples for the validation of predicted accuracy (only) in conjunction with the baseline predicted accuracy processing of the earlier sections of this document. However, the corresponding predicted accuracy validation normalized error tolerances at the 99, 90, and 50% probability-levels (e.g., Tables 5.4.2-2 through 5.4.2-4) must be customized based on the assumed amount of common correlation amongst the error samples in a correlated sub-collection of error samples (e.g., approximately 0.80 positive correlation for each sub-collection, taking into account all contributing major error sources: their expected magnitude and degree of correlation). This is somewhat involved and further assumes that the correlation can be reasonably estimated. However,

this method can be used to "extract" all the information possible from the entire set of error samples – the lower the amount of positive correlation in a correlated sub-collection, the more the additional information available. This can be particularly useful when the number of i.i.d. samples is small to begin with.

For example, with 100 original i.i.d. horizontal error samples, each with an additional 5 correlated samples available with an assumed common correlation of 0.50, it was found (by systematic trial-and-error) that processing all 600 samples using baseline predicted accuracy processing but with normalized error tolerances set to values corresponding to high predicted accuracy fidelity and 400 i.i.d. samples (see Table 5.4.2-2) was applicable. Results are presented in Figure 5.6.3-1 below which basically duplicate the earlier baseline results corresponding to an explicit 400 i.i.d. samples of Figure 5.4.2-1, and are also better (less roll-off or Type II error) than if we had only used the original 100 i.i.d. samples, with earlier baseline results presented in Figure 5.4.2-2.

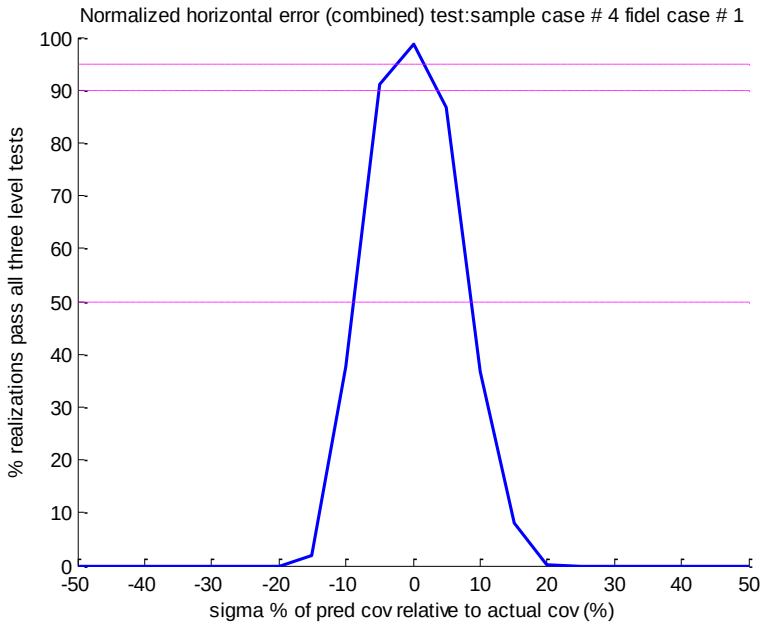


Figure 5.6.3-1: Probability (confidence) of passing all three normalized error tests; high fidelity, 100 i.i.d. samples, each with an additional 5 error samples correlated at 0.50; all 600 individual error samples utilized.

As a reminder, the All Error Samples approach is not suitable for the validation of accuracy (as opposed to predicted accuracy) as it requires i.i.d. samples for the use of order statistics.

5.6.4 Representative Error Samples Approach

This method uses sample statistics to combine all error samples within a correlated sub-collection into one representative i.i.d. error sample and then proceeds with baseline accuracy validation processing as well as baseline predicted accuracy validation processing. In particular, regarding the latter, it uses representative i.i.d. error samples only as well as the corresponding baseline normalized error tolerances. For example, it uses Tables 5.4.2-2 through 5.4.2-4 for normalized error tolerances at the

three probability-levels, where the number of representative i.i.d. error samples (not total number of error samples) is used to index into the tables. Although the number of i.i.d. error samples does not increase with this method, the "entire" collection/extraction solution-space is represented as much as possible.

There are two sub-methods of the representative sample method, each using a somewhat different set of algorithms/equations to generate a representative i.i.d. error sample and normalized representative i.i.d. error sample. They are described, along with examples, in the following Subsections 5.6.4.1 and 5.6.4.2.

Note that regardless the sub-method, the Representative Error Samples approach is only applicable to the validation of (absolute) accuracy and the validation of predicted accuracy. It is not applicable to the validation of relative accuracy or predicted relative accuracy, as the error samples in a correlated sub-collection must be retained for corresponding processing per Section 5.3.

5.6.4.1 Sub-method 1 for Representative Error Samples

Sub-method 1 is the baseline generation method for representative i.i.d. normalized error samples. It is relatively easy to implement and intuitive.

The applicable algorithm/equations for sub-method 1 is presented in Equation (5.6.4.1-1) below and assumes an approximate common diagonal predicted error covariance matrix with standard deviations σ_x and σ_y for each sample $k = 1,..,n$, in a correlated sub-collection and an empirical "combination algorithm" which is approximately equivalent to the sub-method 1.

In the following, the sample mean represents the "common" error between the samples, and the sample sigma or standard deviation about this sample mean represents the "uncorrelated portion" of error in the samples.

- Compute the sample mean and sample standard deviation about the sample mean for each error component over the k samples: $mean\_x$, $sigma\_x$, $mean\_y$, $sigma\_y$ (5.6.4.1-1)
    - both computation of the sample mean and sample standard deviation are unbiased estimates – see Section 5.2.1 of TGD 2b for the explicit calculations
- $eh = ((mean\_x)^2 + (sigma\_x)^2 + (mean\_y)^2 + (sigma\_y)^2)^{\frac{1}{2}}$
- $eh\_norm = \left(((mean\_x)^2 + (sigma\_x)^2)/\sigma_x^2 + ((mean\_y)^2 + (sigma\_y)^2)/\sigma_y^2\right)^{1/2}$
- $\epsilon h\_norm_{99} = \epsilon h\_norm/3.035$, $\epsilon h\_norm_{90} = \epsilon h\_norm/2.146$, $\epsilon h\_norm_{50} = \epsilon h\_norm/1.177$
- test: $\epsilon h\_norm_{99} < 1$, $\epsilon h\_norm_{90} < 1$, and $\epsilon h\_norm_{50} > 1$
- $rep\_pred\_radial_{XX} = \epsilon h\_norm_{XX}/\ \epsilon h$, for plotting
- In addition, $\epsilon h\_norm_{XX} = eh/CE\_XX$, where $XX = 99,90,50$, if scalar accuracy metrics are to be used

Proceed with baseline accuracy and predicted accuracy validation for horizontal errors using the resultant horizontal radial error eh and the normalized horizontal radial errors at the three different

probability levels that correspond to the representative horizontal error sample from each of the correlated sub-collections.

The above corresponds to an individual representative (i.i.d.) normalized radial error sample associated with predicted accuracy validation. The corresponding (internal) computation of representative horizontal radial error eh is associated with accuracy validation as well and corresponds to taking the (approximate) root-mean-square (rms) of the individual correlated samples in the sub-collection.

The use of an (approximate) rms makes sense as it averages "error-power" over the samples. It would be exactly equivalent to the rms if the sample sigma was computed as a biased estimate, i.e., involve the division of n instead of $(n-1)$ samples. However, experimental results for accuracy validation presented in Section 5.6.4.3 are somewhat better using the unbiased estimate. But if samples are few (less than 5) division by n should be used.

Appendix G describes modifications to the above algorithm/equation if the predicted error covariance matrix is not approximately diagonal and common across the correlated samples in a correlated sub-collection. The above algorithm/equation was for 2d horizontal error samples. Similar algorithms/equations are presented in Appendix H for both vertical and 3d error samples.

The following presents a detailed example of the above algorithm/equation for horizontal error samples.

Example

A simulation was performed that generated 60 sub-collections of horizontal error, each sub-collection containing 6 horizontal (2d) error samples correlated at 90% between each pair of samples in the sub-collection.

For each sub-collection, the simulation generated 6 (2d) horizontal error samples from a mean-zero multi-variate Gaussian probability distribution with a 12x12 joint covariance matrix consisting of block diagonals equal to $cov = \begin{bmatrix} 1 & 0 \\ 0 & 1.5625 \end{bmatrix}$ and off-diagonal blocks consisting of $crosscov = 0.9 \cdot cov$, using the same method as detailed in Section 5.6.1.

The matrix cov was also the assumed common predicted error covariance matrix used by the algorithm/equation for all samples in the sub-collection, i.e., $\sigma_x = 1$ and $\sigma_y = 1.25$. This common predicted error covariance matrix is allowed to differ between sub-collections, but did not in this example. Furthermore, since cov and the predicted covariance matrix were identical in this example, it is expected that the validation of predicted accuracy corresponding to high predicted accuracy fidelity will pass with high confidence. In addition, a specified horizontal accuracy requirement of $CE90_{spec} = 3.1$ meters was assumed provided which is approximately 30% larger than the true $\epsilon h_{90} = 2.43$ in this example; therefore, it is expected that validation of accuracy will pass as well with high confidence.

Following generation of the samples, Equation 5.6.4.1-1 was implemented followed by the baseline validation of accuracy (Sections 4.1/5.1), but using representative error samples only instead of all of the

error samples. Each representative error sample was simply selected as the first correlated error sample in the sub-collection for this simple simulation. Corresponding simulated error samples, intermediate variables associated with Equation 5.6.4.1-1, and validation results are presented in Table 5.6.4.1-1.

Table 5.6.4.1-1: Simulation and accuracy validation results corresponding to sub-method 1 of the representative error samples approach and 60 representative error samples

cor sub-collection	1		2		…	60	
error components	x	y	x	y		x	y
cor hor error 1	0.3404	-0.4432	1.1941	0.8697		0.9775	-0.3895
	0.2885	-0.1001	1.8352	1.0641		1.5275	0.5413
…	0.3371	-0.594	1.305	0.6484	…	0.8498	-0.3835
	0.0598	-0.7392	1.9682	0.6259		0.9749	-0.1263
	0.545	-0.3383	1.4509	0.8223		1.4814	-0.139
cor hor error 6	0.0219	-0.0601	1.6481	0.7785		1.4246	0.4877
sample mean	0.2654	-0.3791	1.5669	0.8015		1.206	-0.0015
sample sigma	0.1955	0.2688	0.3035	0.1605	…	0.3031	0.4159
rep hor error	0.3296	0.4647	1.596	0.8174		1.2435	0.4159
rep hor radial error	0.5698		1.7932			1.3112	
best est 90th percentile hor radial error				2.2970 m, where true value 2.4340 m			
lub of 90th percentil hor radial at 90% confidence				2.5430 m, where true value 2.4340 m			

For a given correlated sub-collection in the above table: (1) the sample mean and sample sigma for the x component of error were root-sum-squared to yield the x-component of representative horizontal error, (2) the sample mean and sample sigma for the y component of error were root-sum-squared to yield the y-component of representative horizontal error, and (3) the resultant x-component of representative horizontal error and the resultant y-component of representative horizontal error were root-sum-squared to yield the representative horizontal radial error ϵh of Equation (5.6.4.1-1). Note that representative horizontal error was not identified explicitly in Equation (5.6.4.1-1), but is defined as $\epsilon X = [\epsilon x \quad \epsilon y]^T$, where the components are always positive but have the correct magnitude. These components are also involved in the computation of normalized horizontal radial error (eh_norm) in Equation (5.6.4.1-1).

Baseline validation of predicted accuracy corresponding to high predicted accuracy fidelity (Section 4.2/5.2) was also performed using the same simulation inputs, values of the intermediate variables presented above in Table 5.6.4.1-1, and computation of normalized representative radial errors via Equation 5.6.2.1-1, with results presented in Table 5.6.4.1-2.

Table 5.6.4.1-2: Simulation and predicted accuracy validation result corresponding to sub-method 1 of the representative error samples approach and 60 representative error samples

cor sub-collection	1	2	…	60
norm rep radial errors:				
at prob level 99%	0.1688	0.5859		0.4372
at prob level 90%	0.2387	0.8286		0.6184
at prob level 50%	0.4352	1.5107		1.1275
percent passing:				
prob level 99% < 1 test	98% ; pass as at least 93% req'd			
prob level 90% < 1 test	92% ; pass as at least 79% req'd			
prob level 50% > 1 test	58% ; pass as at least 38% req'd			

As seen in the above tables, both the validation of accuracy and the validation of predicted accuracy were successful. See Appendix G for the corresponding pseudo-code used to generate this example.

Validation success versus sigma deviation

Predicted accuracy validation success vs sigma deviation based on sub-method 1 are also presented in Figure 5.6.4.1-1 below and correspond to 100 representative (i.i.d.) horizontal normalized error samples and high predicted accuracy fidelity; each representative error sample was a combination of an original i.i.d. error sample with an additional 5 error samples correlated at 0.8; i.e., 100 correlated sub-collections containing 6 correlated error samples each, and assumed uncorrelated across sub-collections. (See Appendix G for corresponding pseudo-code used to generate the results.)

Note that it reasonably matches the results of Figure 5.4.2-2 which are based on 100 (non-representative or “original”) i.i.d. error samples. However, the latter will not represent the entire collection solution-space as well as the representative samples.

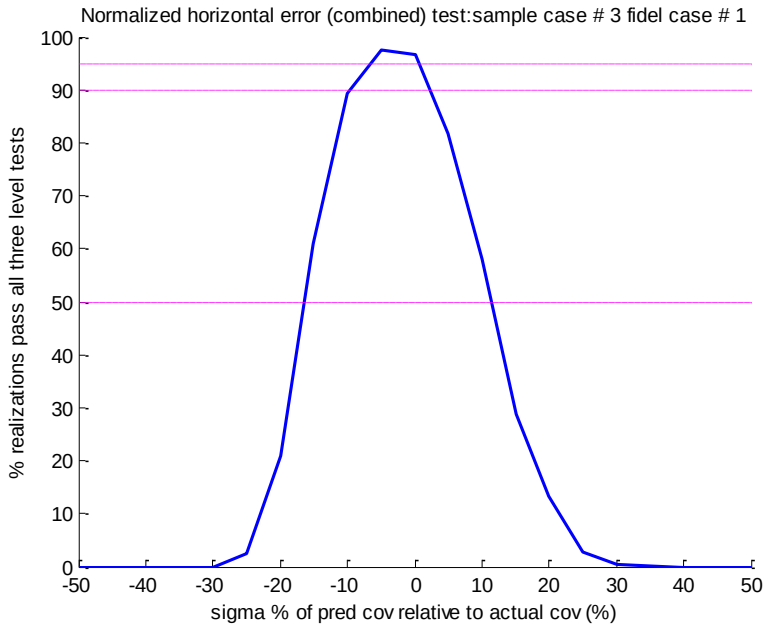


Figure 5.6.4.1-1: Probability of passing all three normalized error tests; high fidelity, 100 representative i.i.d. error samples, each generated from 6 error samples correlated at 0.80; based on Equation (5.6.4.1-1)

The above addressed predicted accuracy and its validation using sub-method 1. Associated accuracy and its validation are also of interest using this sub-method and addressed in Subsection 5.6.4.3.

5.6.4.2 Sub-method 2 for Representative Error Samples

Sub-method 2 for the generation of representative i.i.d. error samples is an alternative to the baseline sub-method 1. It is used in some operational systems.

The applicable algorithm/equations for sub-method 2 is presented in Equation (5.6.4.2-1) below and assumes an approximate common diagonal predicted error covariance matrix with standard deviations σ_x and σ_y for each sample $k = 1,..,n$, in a correlated sub-collection and an empirical "combination algorithm".

In the following, the sample mean represents the "common" error between the samples, and the sample sigma or standard deviation about this sample mean represents the "uncorrelated portion" of error in the samples.

- Compute the sample mean and sample standard deviation about the sample mean for each error component over the k samples: $mean\_x$, $sigma\_x$, $mean\_y$, $sigma\_y$ (5.6.4.2-1)
    - the computation of the sample mean and the sample standard deviation are unbiased estimates – see Section 5.2.1 of TGD 2b for the explicit calculations.

- $eh = ((abs(mean\_x) + 0.5sigma\_x)^2 + (abs(mean\_y) + 0.5sigma\_y)^2)^{1/2}$

- $eh\_norm = \left((abs(mean\_x) + 0.5sigma_x)^2/\sigma_x^2 + \left(abs(mean\_y) + 0.5sigma_y\right)^2/\sigma_y^2\right)^{1/2}$
- $\epsilon h\_norm_{99} = \epsilon h\_norm/3.035$, $\epsilon h\_norm_{90} = \epsilon h\_norm/2.146$, $\epsilon h\_norm_{50} = \epsilon h\_norm/1.177$
- test: $\epsilon h\_norm_{99} < 1$, $\epsilon h\_norm_{90} < 1$, and $\epsilon h\_norm_{50} > 1$
- $rep\_pred\_radial_{XX} = \epsilon h\_norm_{XX}/\ \epsilon h$, for plotting
- In addition, $\epsilon h\_norm_{XX} = eh/CE\_XX$, where $XX = 99{,}90{,}50$, if scalar accuracy metrics are to be used.

The above correspond to an individual representative (i.i.d.) normalized error sample associated with predicted accuracy validation. The corresponding (internal) computation of representative radial error eh is used for accuracy validation as well. Note that when correlation between samples in a correlated sub-collection is small, the sample standard deviation or "sigma" dominates the absolute sample mean-value (mean-value approaches zero), and since only one-half of the sigma value is used in the above equation, eh will be typically too small as will eh_norm.

Predicted accuracy validation success vs sigma deviation based on sub-method 2 are presented in Figure 5.6.4.2-1 below and correspond to 100 representative (i.i.d.) horizontal normalized error samples and high predicted accuracy fidelity; each representative error sample was a combination of an original i.i.d. error sample with an additional 5 error samples correlated at 0.8; i.e., 100 correlated sub-collections containing 6 correlated error samples each, and assumed uncorrelated across sub-collections. (See Appendix G for corresponding pseudo-code used to generate the results.)

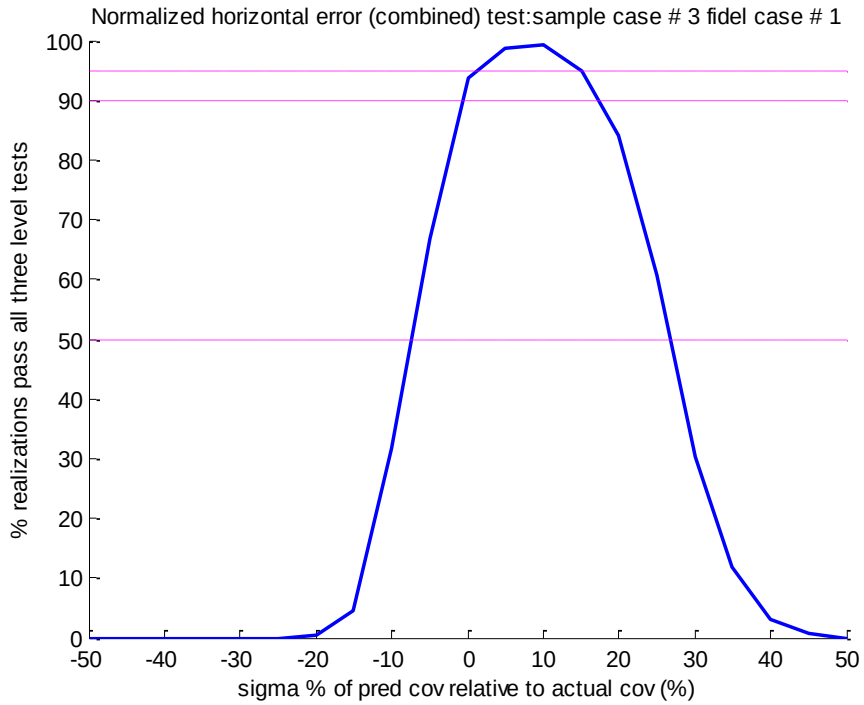


Figure 5.6.4.2-1: Probability of passing all three normalized error tests; high fidelity, 100 representative i.i.d. error samples, each generated from 6 error samples correlated at 0.80; based on Equation (5.6.4.2-1)

Note that results do not match the results of Figure 5.4.2-2 very well which are based on 100 (non-representative or "original") i.i.d. error samples. However, this is sensitive to the amount of correlation, as can be deduced from the results presented in the following subsection. The following subsection also addresses accuracy and its validation using sub-method 2.

5.6.4.3 Comparison of the Sub-methods for Generation of Representative Error Samples

In terms of preference, sub-method 1 (the baseline) is first, followed by sub-method 2. Direct comparison of the sub-methods (aka sub-options) is presented in Figure 5.6.4.3-1 below corresponding to: The percent of representative i.i.d. samples passing the 90% probability normalized error test with at least 90% confidence (perfect predicted accuracy fidelity is assumed and 500 realizations used). Results are indicative of predicted accuracy validation (one of three tests) and are a function (x-axis) of the amount of correlation between samples in a correlated sub-collection. 100 representable samples are assumed, each generated from 6 correlated samples.

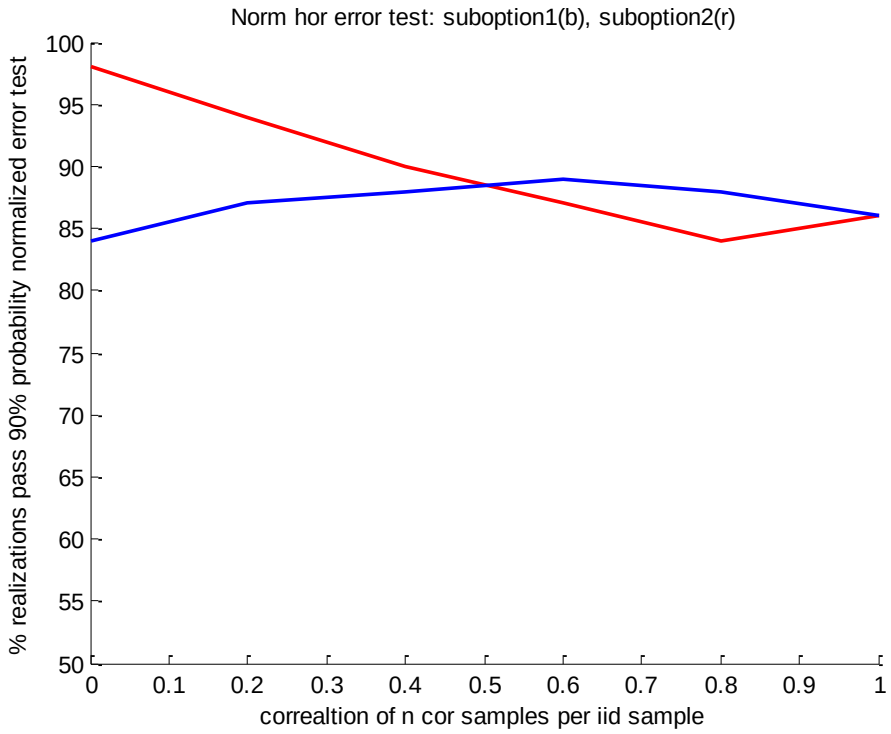


Figure 5.6.4.3-1: Normalized error test results versus correlation (6 correlated samples)

Ideally, the percent should be about 86 if it were to match 100 (non-representative) error sample results exactly (see Table 5.4.1-1), but a percent up to 90% or a little more is certainly reasonable. This is true for sub-method 1, but only true for certain correlation ranges for sub-method 2 per Figure 5.6.4.3-1.

Additional results are presented in Figures 5.6.4.3-2 and 5.6.4.3-3 for a different number of correlated samples per correlation sub-collection. Sub-method 2 is very sensitive to the assumed amount of correlation; whereas sub-method 1 is only sensitive when there are a small number of correlated samples per correlated sub-collection (Figure 5.6.4.3-3). Regarding the latter, sensitivity could be

reduced by computing the sample sigma as a biased estimate (computation involves dividing by the number of correlated samples n instead of (n-1)). This would decrease the representative radial error; hence, reduce the normalized error and therefore increase the "pass rate". This would raise the blue curve for lower values of correlations where the sample sigma dominates relative to the value of the sample mean (see Equation 5.6.4.1-1). However, it also degrades somewhat the accuracy validation results documented later in this subsection.

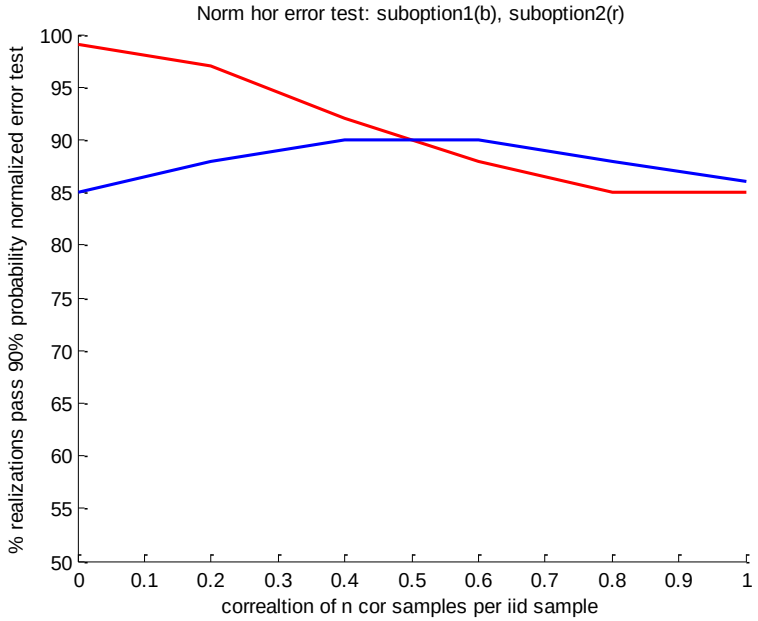


Figure 5.6.4.3-2: Normalized error test results versus correlation (16 correlated samples)

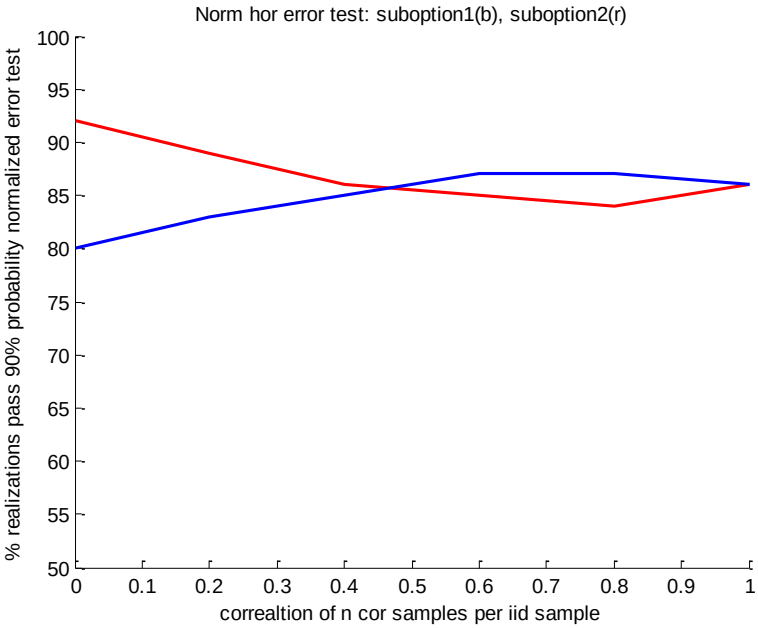


Figure 5.6.4.3-3: Normalized error test results versus correlation (3 correlated samples)

In addition, accuracy validation results (as opposed to predicted accuracy validation results related to the above) were also computed, comparing those obtained using sub-methods 1 and 2, versus using 100 (original) i.i.d. samples as a "truth". Order statistics were applicable for all methods and used to compute both the least-upper-bound for the 90th horizontal percentile at a 90% confidence-level as well as a best estimate of the percentile. Results were averaged over 500 realizations and are presented in Table 5.6.4.3-1 below. As can be seen, sub-method 1 provided the best overall results relative to using original i.i.d. samples. However, results between the two sub-methods were generally similar for higher values of correlation.

See Appendix G for corresponding pseudo-code used to generate the results of Table 5.6.4.3-1. Errors were simulated consistent with a true 90th horizontal percentile equal to 3.47 meters which closely matches the "original iid be" results presented in the table.

Table 5.6.4.3-1: Horizontal radial error 90th percentile least-upper-bound (lub) and best estimate (be) for 100 representative errors sub-methods 1 and 2 and "truth" corresponding to 100 original i.i.d. error samples (orange: poor results; yellow: use with caution)

number cor samples per correlated sub-collection = 3						
cor =	sub-method 1 lub	sub-method 2 lub	orig iid lub	sub-method 1 be	sub-method 2 be	orig iid be
0	4.24	3.52	4	3.73	3..13	3.43
0.2	4.02	3.71	3.96	3.58	3.29	3.4
0.4	3.9	3.9	3.99	3.47	3.43	3.42
0.6	3.83	4.03	3.98	3.4	3.54	3.44
0.8	3.85	4.12	3.97	3.36	3.59	3.43
0.999	3.99	4.02	3.99	3.43	3.46	3.43
number cor samples per correlaed sub-collection = 6						
cor =	sub-method 1 lub	sub-methods 2 lub	orig iid lub	sub-mehod 1 be	sub-method 2 be	orig iid be
0	4.06	2.95	3.99	3.54	2.63	3.42
0.2	3.87	3.35	3.99	3.41	2.98	3.42
0.4	3.75	3.67	3.99	3.36	3.23	3.42
0.6	3.71	3.9	4	3.31	3.43	3.44
0.8	3.8	4.07	3.99	3.32	3.55	3.43
0.999	4.01	4.04	4.01	3.43	3.46	3.43
	number of samples per correlated sub-collection = 16					
cor =	sub-method 1 lub	sub-method 2 lub	orig iid lub	sub-method 1 be	sub-method 2 be	orig iid be
0	4.02	2.51	4	3.47	2.22	3.44
0.2	3.8	3.1	3.99	3.33	2.75	3.44
0.4	3.66	3.51	3.98	3.27	3.1	3.43
0.6	3.61	3.77	3.97	3.22	3.32	3.42
0.8	3.75	4.01	3.99	3.26	3.48	3.43
0.999	3.99	4.02	3.99	3.49	3.47	3.45

In summary, taking both accuracy and predicted accuracy validation into account, sub-method 1 is preferred for the Representative Error Samples approach to account for the (positive) correlation between the samples in a correlated sub-collection.

5.7 The Effect of Errors in Ground Truth Data and Corresponding Compensation

Errors in the coordinate values of ground truth contribute to sample errors since geolocations are differenced from corresponding ground truth locations in order to generate these sample errors. The validation processes described in the earlier sections of this document assumed that this was not an issue – ground truth accuracy was assumed much better than the corresponding NSG system's geolocation accuracy under test; hence, their contributory errors were negligible. This section of the document presents guidance regarding when this assumption is actually true, and how processing should change when errors in ground truth coordinate values cannot be dismissed. It also presents practical advice, i.e., "bends" theory somewhat for real-world validation when the ideal is seldom applicable.

(Note: the term "ground truth" can be considered somewhat of a misnomer as actual ground truth points are in error; hence, some prefer the term "reference point" instead [4].)

Assume that (*a priori*) ground truth accuracy is expressed as a scalar accuracy metric such as $CE90_{GT}$ meters. Furthermore, assume that it is expressed as a percentage (unit-less) of the corresponding accuracy under test, assumed equal to $CE90_{spec}$ since its actual value is not known. Their combined or total test accuracy is their root-sum-square, again assumed expressed as a percentage of the corresponding accuracy under test and illustrated in Figure 5.7-1 below:

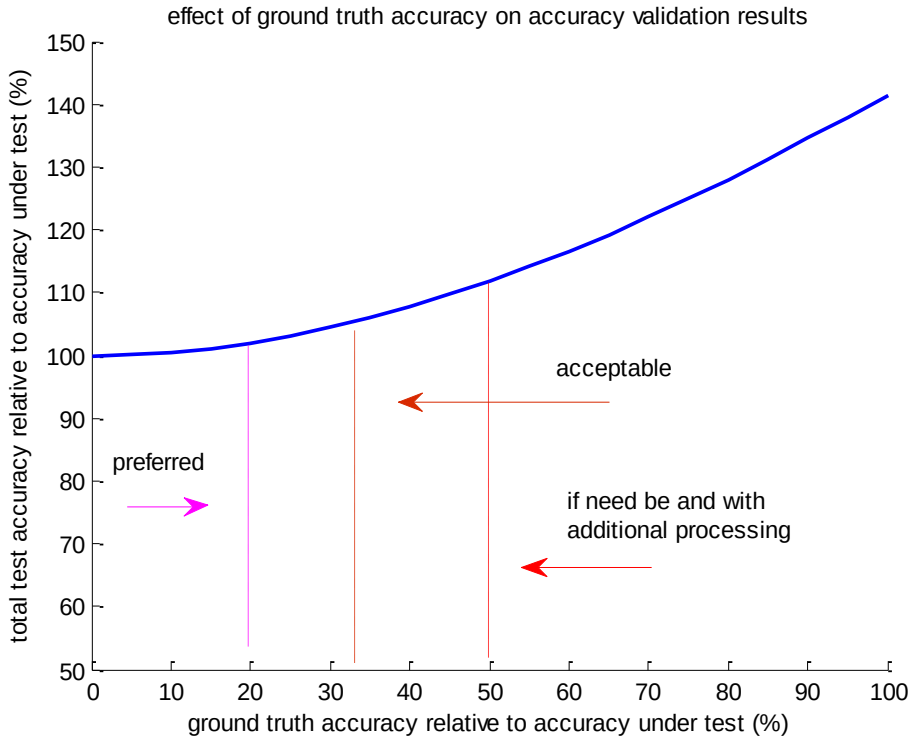


Figure 5.7-1: The effect of ground truth accuracy on total test accuracy

Thus, for example, if ground truth accuracy is equal to 20% of the accuracy under test ("preferred" in Figure 5.7-1), total test accuracy is only approximately 2% larger than the accuracy under test. The following is recommended guidance on validation processing as a function of ground truth accuracy expressed as a percentage of the accuracy under test:

(1) If 20% or less: proceed as usual for both the validation of accuracy and the validation of predicted accuracy – no special processing is required.

This is the preferred and recommended level for ground truth accuracy. As arbitrary but specific examples, if addressing the validation of an NSG geolocation system with a 2 meter accuracy requirement, ground truth accuracy should be 40 centimeters or better; if a 5 meter accuracy requirement, 1 meter or better.

It is also of benefit to include this "20% or less" guidance in the formal specification of accuracy requirements as part of the general description of corresponding validation requirements.

Note that reference [1] recommends "10% or less", i.e., "an order of magnitude better"; however, although desirable, particularly if ground truth error includes a considerable bias, this requirement is considered too stringent for most practical validations and is to be considered a "goal" instead.

(2) If 33% or less but larger than 20%: proceed as usual for the validation of accuracy assuming that there is at least a 10% margin built-in to the specified accuracy requirement for validation (not unusual); proceed as usual for the validation of predicted accuracy other than that corresponding to high predicted accuracy fidelity. (See [4, 6] for additional references regarding the applicability of 33% or the "3:1" rule.)

(3) If the above are not applicable because the *a priori* ground truth accuracy is relatively too large and if better ground truth cannot be obtained, (reluctantly) proceed as follows:

> **(a) If ground truth accuracy is no larger than 50% the value of the accuracy under test:** proceed with modified validation processing as outlined in the next paragraphs.
>
> **(b) If ground truth accuracy is greater than 50% the value of the accuracy under test:** formal validation should not be performed.

Modified Accuracy Validation Processing

The following assumes that horizontal and vertical radial 90th percentiles are of interest and the corresponding specified accuracy requirements are expressed as $CE90_{spec}$ and $LE90_{spec}$, respectively, and the ground truth accuracy estimates are expressed as $CE90_{GT}$ and $LE90_{GT}$, respectively.

Use the following adjusted requirements during validation and then proceed as usual:

$$CE90_{spec\_adj} = \sqrt{CE90_{spec}^2 + CE90_{GT}^2} \qquad (5.7\text{-}1)$$

$$LE90_{spec\_adj} = \sqrt{LE90_{spec}^2 + LE90_{GT}^2}$$

Modified Predicted Accuracy Validation Processing

For the normalized error tests, first adjust the predicted error covariance matrix for sample k by the corresponding *a priori* error covariance for the ground truth (the same ground truth covariance is assumed for all samples) and then proceed as usual:

$C_X\_pred_{s\_k} \rightarrow C_X\_pred_{s\_k} + Cov_{GT}$, where (5.7-2)

$$Cov_{GT} = \begin{bmatrix} \left(CE90_{GT}/d_{h\_90}\right)^2 & 0 & 0 \\ 0 & \left(CE90_{GT}/d_{h\_90}\right)^2 & 0 \\ 0 & 0 & \left(LE90_{GT}/d_{v\_90}\right)^2 \end{bmatrix} \text{and}$$

$d_{h\_90} = 2.146$ and $d_{v\_90} = 1.645$. (The CE scale factor d_{h_90} is based on assumed and approximate value of 1 for the "sqrt_ratio_eigenvalue" corresponding to the actual Cov_{GT}, i.e., a non-elongated error ellipse, which is a reasonable assumption for accurate surveyed ground truth.)

All of the above processing assumed the use of $CE90$ and $LE90$ for convenience. Corresponding guidance is also applicable to (3d) radial errors (accuracy) as well as to different levels of probability (95% or 50%) in a straight-forward and similar manner.

Correlated Errors

The above assumed that correlated ground truth errors were not a factor. This is certainly true when ground truth errors are negligible. When not:

If the above modified validation processing is applicable, any intra-state correlations between a ground point's components of errors (e.g. ϵx correlated with ϵy), if known, can be incorporated directly as off-diagonal terms in the assumed Cov_{GT}, although typically not required.

If the above modified validation processing is applicable, any inter-state correlation (correlation of errors between different ground points) for ground points corresponding to a given set of sensor data (but not across different sets) will automatically be addressed per the techniques of Section 5.6

If correlation is across different sets (e.g., same test site used for many different sets of sensor data), there is no practical modification of validation processing. However, although not ideal, the effect can be ignored because ground truth accuracy is assumed no greater than 50% the value of the accuracy under test; thus, correlation corresponding to combined errors between different error samples is no greater than (plus) 20% assuming independent error samples for the other contributing errors. This amount of correlation can be ignored for reasonable but approximate validation results. However, if ground truth accuracy is greater than 50% the value of the accuracy under test, then this correlation cannot be ignored and formal validation should not be done (another reason for the earlier 50% restriction.)

(Note: the above 20% correlation between different samples of combined errors is derived as follows: $cor \leq \frac{0.5^2}{1+0.5^2} = 0.20$.)

Relative Accuracy

In general, all of the previous guidance of the current section is also applicable to relative accuracy and predicted relative accuracy. However, subsequent requirements on ground truth point accuracy may be more stringent (as a relative percent) since requirements for relative accuracy may be more stringent than for (absolute) accuracy. On the other hand, validation of relative accuracy involves only point-pairs extracted from the same set of sensor data. Hence, any correlation of corresponding ground truth errors is beneficial as they tend to cancel out statistically. In particular, a relevant pair of correlated ground truth points has a combined effect of relative accuracy equal to $\sqrt{2(1-cor)}\,CE90_{GT}$, for example. The higher the (positive) correlation, the less the effect of ground truth errors on the validation of relative accuracy.

6 Notes

6.1 Intended Use

This information and guidance document provides technical guidance to inform the development of geospatial data accuracy characterization for NSG GEOINT collectors, producers and consumers -- accuracy characterization as required to describe the trustworthiness of geolocations for defense and intelligence use and to support practices that acquire, generate, process, exploit, and provide geolocation data and information based on geolocation data. This document is part of a series of complementary documents. TGD 2c provides technical guidance for methods, practices, and algorithms in specification and validation as of part of a series of information and guidance documents titled Accuracy and Predicted Accuracy in the NSG. Other documents in this series address a more generalized overview of accuracy and predicted accuracy and additional topic specific technical guidance in predictive statistics, sample statistics, estimators and quality control, Monte-Carlo simulation, and external data and quality assessment.

7 References

[1] Ager, T., “An Analysis of Metric Accuracy Definitions and Methods of Computation, NGA white paper, 2004.

[2] Bar-Shalom, Yaakov and Thomas Fortmann, Tracking and Data Association, Academic Press, 1988.

[3] Beekman, J., et al; TLE Evaluation Methodology, Rev. 12, USAF ACC/USAFWC/53WG/53TMG/59th TES, Nellis AFB, NV, April 2016.

[4] Congalton, R.G. and K. Green, Assessing the Accuracy of Remotely Sensed Data: Principles and Practices, Second Edition, CRC Press, 2009.

[5] Dolloff, John, and Carr, Jacqueline, " Methods for the specification and validation of geolocation accuracy and predicted accuracy ", Proc. SPIE 10199, Geospatial Informatics, Fusion, and Motion Video Analytics VII, 1019908 (May 1, 2017); doi:10.1117/12.2263855; http://dx.doi.org/10.1117/12.2263855, last accessed 30 May 2017.

[6] Dolloff, John, and Settergren, R., “An Assessment of WorldView-1 Positional Accuracy based on Fifty Contiguous Stereo Pairs of Imagery”, PE&RS, Vol. 76, No, 8, August 2010.

[7] Dolloff, J. T., and Theiss, H. J., The Specification and Validation of Predicted Accuracy Capabilities for Commercial Satellite Imagery, Proceedings of the ASPRS 2014 Annual Conference, https://www.asprs.org/a/publications/proceedings/Louisville2014/Dolloff.pdf, last accessed 30 May 2017.

[8] Dolloff, J.T., and Theiss, H.J., “Temporal Correlation of Metadata Errors for Commercial Satellite Images: Representation and Effects of Stereo Extraction Accuracy”, ISPRS XXII Congress, 2012, http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXIX-B1/215/2012/isprsarchives-XXXIX-B1-215-2012.pdf, last accessed 30 May 2017.

Appendix A: Additional Terms and Definitions

There are a number of authoritative guides as well as existing standards within the NSG and Department of Defense for definitions of the identified additional terms used in this technical guidance document. In many cases, the existing definitions provided by these sources are either too general or, in some cases, too narrow or dated by intended purposes contemporary to the document's development and publication. The definitions provided in this document have been expanded and refined to explicitly address details relevant to the current and desired future use of accuracy in the NSG. To acknowledge the basis and/or linage of certain terms, we reference the following sources considered as either foundational or contributory:

[a] Anderson, James M. and Mikhail, E., Surveying: Theory and Practice, 7th Edition, WCB/McGraw-Hill, 1998.

[b] DMA-TR-8400.1, DMA Technical Report: Error Theory as Applied to Mapping, Charting, and Geodesy.

[c] Defense Mapping Agency, Glossary of Mapping, Charting, and Geodetic Terms, 4th Edition, Defense Mapping Agency Hydrographic/Topographic Center, 1981.

[d] ISO TC/211 211n2047, Text for ISO 19111 Geographic Information - Spatial referencing by coordinates, as sent to the ISO Central Secretariat for issuing as FDIS, July 17, 2006.

[e] Joint Publication (JP) 1-02, Department of Defense Dictionary of Military and Associated Terms, November 8, 2010 as amended through January 15, 2016.

[f] MIL-HDBK-850, Military Handbook: Glossary of Mapping, Charting, and Geodetic Terms, January 21, 1994.

[g] MIL-STD-2401, Department of Defense Standard Practice; Department of Defense World Geodetic System (WGS), January 11, 1994

[h] MIL-STD-600001, Department of Defense Standard Practice; Mapping, Charting and Geodesy Accuracy, February 26, 1990.

[i] *National System for Geospatial Intelligence* [Brochure] Public Release Case #15-489.

[j] NGA.STND.0046_1.0, The Generic Point-cloud Model (GPM): Implementation and Exploitation, Version 1.0, October 03, 2015.

[k] Oxford Dictionaries (www.oxforddictionaries.com/us/) copyright © 2016 by Oxford University Press.

[l] Soler, Tomas and Hothem, L., "Coordinate Systems Used in Geodesy: Basic Definitions and Concepts", Journal of Surveying Engineering, Vol. 114, No. 2, May 1988.

A priori - Relating to or denoting reasoning or knowledge that proceeds from theoretical deduction rather than from observation or experience. [k]

- For typical NSG accuracy and predicted accuracy applications, *a priori* refers to a mathematical statistical model of errors and/or the corresponding state vector containing those errors prior to its adjustment using additional information.

A posteriori - Relating to or denoting reasoning or knowledge that proceeds from observations or experiences to the deduction of probable causes. [k]

- For typical NSG accuracy and predicted accuracy applications, *a posteriori* refers to a refined mathematical statistical model of errors and/or the corresponding state vector containing those errors following its adjustment using additional information.

Absolute Horizontal Accuracy - The range of values for the error in an object's horizontal metric geolocation value with respect to a specified geodetic horizontal reference datum, expressed as a radial error at the 90 percent probability level (CE). [b],[f],[j]

- There are two types of absolute horizontal accuracy: *predicted* absolute horizontal accuracy is based on error propagation via a statistical error model; and *measured* absolute horizontal accuracy is an empirically derived metric based on sample statistics.
- The term "horizontal accuracy" is assumed to correspond to "absolute horizontal accuracy".
- The 90% probability level (CE) is the default; 95% and 50% probability levels are optional, i.e., CE_95 and CE_50, respectively.

Absolute Vertical Accuracy - The range of values for the error in an object's metric elevation value with respect to a vertical reference datum, expressed as a linear error at the 90 percent probability level (LE). [b],[f],[j]

- There are two types of absolute vertical accuracy: *predicted* absolute vertical accuracy is based on error propagation via a statistical error model; and *measured* absolute vertical accuracy is an empirically derived metric based on sample statistics.
- The term "vertical accuracy" is assumed to correspond to "absolute vertical accuracy".
- The 90% probability level (LE) is the default; 95% and 50% probability levels are optional, i.e., LE_95 and LE_50, respectively.

Bias Error - A category of error; an error that does not vary from one realization (trial or experimental outcome) to the other. When error is represented as a random variable, random vector, stochastic process, or random field, a bias error corresponds to a non-zero mean-value. [f],[j]

- Caution: a given realization of a mean-zero stochastic process with typical temporal correlation and over a reasonable finite time interval appears to have a non-zero sample mean-value; however, when sample statistics are taken over enough multiple (independent) realizations, the sample mean-value approaches zero in accordance with the true mean-value. This characteristic extends to random fields as well.

Confidence Ellipsoid - An ellipsoid centered at an estimate of geolocation such that there is a 90% probability (or XX% if specified specifically) that the true geolocation is within the ellipsoidal boundary (ellipsoid interior). A confidence ellipsoid is typically generated based on an error covariance matrix, an assumed mean-value of error equal to zero, and an assumed multi-variate Gaussian probability distribution of error in up to three spatial dimensions.

Confidence Interval - A type of interval estimate of an unknown population parameter in statistics. More specifically, if X is a vector of random samples from a probability distribution with statistical parameter θ which is to be estimated with confidence-level (confidence coefficient) γ:

- $prob\{a(X) < \theta < b(X)\} = \gamma$, where $a(X)$ and $b(X)$ are random end-points and functions of X.
- Note that the probability distribution need not be specified, but typically is, e.g., a Gausssian (Normal) distribution, a commonly assumed continuous probability distribution.
- Typical parameters represented by θ are the distribution's (or corresponding random variable's) mean-value, standard deviation, or percentile.
- The above confidence interval is a two-sided confidence interval; a one-sided confidence interval involves only $a(X)$ or $b(X)$ and is bounded on one side, e.g., $prob\{\theta < b(X)\} = \gamma$.

Confidence Interval (Order Statistics) - Similar to a confidence interval (see above) except computed using order (nonparametric) statistics in which a specific probability distribution is not assumed and a finite set of ordered (by ascending magnitude) samples y_i, $i = 1, .., n$, of a random variable x are available. Thus, assuming that the unknown probability distribution is continuous and that the parameter of interest is x_p, the p percentile of the corresponding random variable x, as typically the case for an NSG application:

- $prob\{y_k < x_p < y_{k+r}\} \geq \gamma$, where the specific order sample indices k and $(k + r)$ are such that r is the smallest positive integer such that the probability or confidence bound is met.
- Note that once the order sample indices k and $(k + r)$ are specified (determined), the two-sided confidence interval corresponds to a specific confidence $\gamma_0 \geq \gamma$, i.e., $prob\{y_k < x_p < y_{k+r}\} = \gamma_0$. This is the reason why γ is sometimes referred to as the "specified (minimum) confidence-level".
- A one-sided confidence interval based on order statistics is typically of the form $prob\{x_p < y_{k^\wedge}\} \geq \gamma$, where the ordered sample $y_{k^\wedge}$ is the smallest valued ordered sample (equivalently, $k^\wedge$ the smallest index) such that the probability or confidence bound is met.
- Note that all of the above are applicable to discrete probability distributions as well; simply substitute $\leq$ for $<$ inside the interval, for example: $prob\{y_k \leq x_p \leq y_{k+r}\} \geq \gamma$.

Correlated Error - A category of errors; errors that are correlated with other errors, and typically represented in the NSG as a random vector, stochastic processes, or random field. A correlated error is independent (uncorrelated) with itself and other errors from one realization (trial or experimental outcome) to the next. However, within a given realization, it is correlated with other errors of interest:

- If a random vector, the various elements (random variables) which make it up are correlated with each other (intra-state vector correlation).
- If a stochastic process, the collection of random vectors which make up the stochastic process are correlated with each other (inter-state vector correlation). That is, the elements of one random vector are correlated with the elements of another random vector, typically the closer the two random vectors in time, the greater the correlation. A similar concept is applicable to random fields.

Correlated Values - Values (of random variables) which are related by a statistical interdependence. For two random variables, this interdependence is represented by their covariance and typically expressed as a correlation coefficient – both have non-zero values. This interdependence is relative to deviations about their respective mean-values. [f]

Covariance - A measure of the mutual variation of two random variables, where variations (deviations or dispersions) are about their respective mean-values. If the covariance between two random values is zero, they are uncorrelated. [b]

Covariance Matrix - A symmetric, nxn positive definite matrix populated with the variances and covariances of the random variables contained within a single, multi-component ($nx1$) state vector or random vector. Note that if row i ($1 \leq i \leq n$) and all corresponding columns j ($1 \leq j \leq n$,$j \neq i$) are zero, random variable i is uncorrelated with all of the other random variables j. [b]

Cross-covariance Matrix - An nxm matrix containing the covariance between each pair of elements (random variables) of an $nx1$ random vector and an $mx1$ random vector.

Error (augmented definition) - The difference between the observed or estimated value and its ideal or true value. [f] There are a number of different categories of errors applicable to the NSG: Bias Error, Random Error, and Correlated Error. In general, an error of interest may be a combination of errors from these categories. Their combination is typically represented as either a random variable, random vector, stochastic process, or random field:

- A random variable represents a bias error plus a random error. The former corresponds to the random variable's mean-value, and if equal to zero, the random variable represents random error only, which is uncorrelated from one realization of the random variable to the next realization.
- A random vector, stochastic process, and random field can represent all three categories of error. The random variables that make-up (are elements of) random vectors are uncorrelated from one realization to the next by definition. However, within a given realization, they can also be correlated with each other:
    - For a random vector per se, this correlation is also termed "intra-state vector correlation".
    - For a stochastic process, which consists of a collection of random vectors, random variables in one random vector can also be correlated with random variables in another random vector; this is also termed "inter-state vector" correlation. The same concept is applicable to random fields.

Error Ellipsoid - An ellipsoid such that there is a 90% probability (or XX% if specified specifically) that geolocation error is within the ellipsoidal boundary (ellipsoid interior). An error ellipsoid can be generated based on a predictive or sample-based error covariance matrix, centered at an assumed predictive mean-value of error equal to zero or a sample-based mean-value of error not equal to zero, and an assumed multi-variate Gaussian probability distribution of error in up to three dimensions.

Estimator - An algorithm/process which estimates the value of an nx1 state vector. Its inputs are measurements related to the state vector and may include *a priori* information about the state vector.

- An estimator is usually designed to be an optimal estimator relative to a cost function, such as the sum of weighted *a posteriori* measurement residuals, minimum mean-square solution error, etc.
- Estimators are sequential or batch processes, and an optimal estimator should include both an estimate of the state vector and its predicted accuracy, usually an error covariance matrix, as output. A properly implemented MIG for a target's geolocation is an optimal estimator.

Gaussian (or Normal) probability distribution - A specific type of probability distribution for a random variable. The distribution is specified by either a Gaussian probability density function or a Gaussian cumulative distribution function. These in turn are completely characterized by the random variable's mean-value and variance.

- The Gaussian (probability) distribution is a common distribution that approximates many kinds of errors of interest to the NSG, and approximates the distribution for a sum of errors from different (non-Gaussian) distributions as well (Central Limit Theorem). A Gaussian distribution corresponding to an nx1 random vector is termed a multi-variate Gaussian distribution.

Horizontal Error - As applied to geospatial measurements and processes, horizontal error is typically observed in the x, y plane of a local right-handed coordinate system where the x, y plane is defined as tangent to the defined reference surface at the point of origin. While horizontal error is the x and y components of error, it may be generalized by its magnitude or 2D radial error.

Inter-state vector correlation - The correlation between the errors (random variables) of the elements in two different state vectors.

Intra-state vector correlation - The correlation between the errors (random variables) of different elements in the same state vector.

Least-upper-bound - The smallest value (real number) greater than or equal to a quantity of interest, assuming an analytic (deterministic) application. If a statistical or probabilistic-based application, the definition is extended to account for random variables. In particular, for the specification and validation of accuracy based on order statistics using i.i.d. samples of the random variable defined as geolocation radial error, and assuming that horizontal radial error at the 50th percentile is of interest for specificity:

- The least-upper-bound $lub\_\epsilon h_{50}$ is defined as the smallest value greater than the true (and unknown) 50th percentile of horizontal radial error ϵh_{50}, where horizontal radial error ϵh is a random variable. The least-upper-bound also corresponds to an accompanying and independently specified (minimum) level-of-confidence, typically 90%. Correspondingly, $lub\_\epsilon h_{50}$ is equal to the smallest order sample value of horizontal radial error such that:
  - $prob\{\epsilon h_{50} < lub\_\epsilon h_{50}\} \geq 0.90$
- As the number of samples increase, the closer (statistically) $lub\_\epsilon h_{50}$ comes to ϵh_{50}.
- The (probabilistic) lub is equivalent to a one-sided confidence interval based on order statistics.

Local Tangent Plane Coordinate System - A local X,Y,Z right-handed rectangular coordinate system such that the origin is any point selected on a given reference ellipsoid, its XY plane is tangent to the reference ellipsoid at the point of origin, and the Y-axis is typically directed to the North Pole (an East-North-Up (ENU) system). [a]

Mean-Value - The expected value of a random variable. Given a collected sample of measurements, the sample mean-value is the average of the values of the sample measurements. The mean-value of a predictive error is typically assumed zero unless specifically stated otherwise. If correctly modelled, the predictive mean-value should be closely approximated by the sample mean-value taken over a large number of independent and identically distributed samples.

- The concept of mean-value readily extends to random vectors and is the vector of the mean-values of the individual components or random variables making up the random vector. It readily extends to stochastic processes and random fields as well, since they are collections of random vectors. If (wide-sense) stationary or (wide-sense) homogeneous, respectively, their corresponding mean-value is a constant random vector mean-value.

Multi-Image Geopositioning (MIG) - An optimal solution for a "target's" geolocation (state vector) with reliable predicted accuracies based on the (weighted) measurements of the geolocation in one or more images. A batch process which minimizes the sum of weighted *a posteriori* measurement residuals, where the latter may also include measurements equivalent to *a priori* estimates of geolocation. MIG can also correspond to the simultaneous solution for the geolocation of multiple targets. In general, a MIG solution's predicted accuracies correspond to or are derived from the solution's *a posteriori* error covariance matrix.

Order Statistics - Nonparametric statistics performed on a set ordered by ascending magnitude of randomly sampled values. Nonparametric statistics assume no *a priori* information about the underlying probability distribution of a random variable such as its mean-value, variance, or type of probability distribution function. In the NSG, order statistics are used to compute scalar accuracy metrics from independent and identically distributed samples of error.

Percentile - If a random variable's probability (or sample) distribution is divided into 100 equal parts, the value of the random variable that corresponds to the percentage of the distribution equal to or below the specified percentile, e.g. the 90th percentile indicates the lowest sample value such that it is greater than the values of 90 percent of the samples.

- A more formal definition is as follows: The p percentile of a random variable x is defined as the smallest number x_p such that $p = prob\{x \leq x_p\}$. Thus, the probability distribution function (typically unknown) of the random variable x evaluated at x_p is equal to p. x_p is a deterministic parameter with typically unknown value.

Precision - The closeness to one another of a set of repeated observations of a random variable. [a],[f]

- In terms of accuracy, precision is a measure of the repeatability of the underlying errors. High accuracy implies high precision, but not vice versa. For example, for an error represented as a random variable, high precision implies a small standard deviation, but high accuracy implies both a small standard deviation and a small or zero mean-value (or bias).

Probability density function (pdf) - A function that defines the probability distribution of a random variable. If continuous, its integral is the (cumulative) probability distribution function.

Probability distribution - Identifies the probability of a random variable's values over an applicable range of values. There are many different types of probability distributions: Gaussian or Normal, uniform, exponential, etc.

- In most NSG applications for accuracy and predicted accuracy, the random variable and its probability distributions are assumed continuous.
- The probability distribution is specified by either a probability density function or a (cumulative) probability distribution function; either based on an a *priori* model or sample statistics.

Probability distribution function (cdf) - The (cumulative) probability distribution function defines the probability that a random variable's value is less than or equal to a specified number in the interval [0,1].

Radial Error - A generalization of two horizontal error components (x, y) or three dimensional (horizontal and vertical error components – x, y, z) error components to a distance value (magnitude) as measured along the radius of a circle or sphere, respectively.

Random Error - A category of error; a measure of deviation from an ideal or true value which results from an accidental and unknown combination of causes and varies from one measurement to the next. Not deterministic. For NSG applications, a random error is typically represented as a random variable, random vector, stationary process, or random field. And more specifically, as deviations about their mean-values, the latter considered biases. [b],[f]

- The random error corresponding to a random variable or the random error corresponding to (the elements of) a random vector are independent (uncorrelated) from one realization to the next, by definition.
- The random error corresponding to (the elements of) a random vector can also be correlated between the various elements for a given realization (intra-state vector correlation); hence this error is also a correlated error.
- The random error corresponding to a stochastic process corresponds to the collection of random errors associated with the collection of random vectors making up the stochastic process. Random error is independent (uncorrelated) from one realization to the next. However, within a specific realization, the individual random error vectors are typically temporally correlated amongst themselves (inter-state vector correlation); hence, random error is also correlated error. This same characteristic extends to random fields.
- The probability distribution for a random variable representing a random error is arbitrary – not necessarily Gaussian.

Random Error Vector - An error represented by a nx1 random vector, and in the NSG, typically corresponds to the error in a state vector's value. The error itself could correspond to a combination of errors from different error categories: bias error, random error, and/or correlated error. That is, the term "random error vector" does not imply the corresponding category of error is necessarily (only) "random error".

Random Variable - A variable whose value varies by chance, i.e., non-deterministic. Somewhat more formally, a random variable is a mapping from the space of experimental outcomes to a space of numbers. In the NSG, when error is represented by a random variable (a random vector with one component or element, i.e., n=1), its corresponding statistics are specified by a statistical error model.

- For most NSG applications, the space of experimental outcomes is already a number. For example, the x-component of sensor position can be considered a random variable. Equivalently, it can be defined as the true x-component of sensor position plus x-component of sensor position error, the former a deterministic (typically unknown) value and the latter a random variable.
- A random variable is statistically characterized by its mean-value, variance, and (more completely) its probability density function (pdf). The probability density function (pdf) is typically unknown and not included, but if needed for the calculation of probabilities, assumed Gaussian distributed with the pdf completely characterized by the mean-value and variance.

Random Vector - A random vector (RV) is an nx1 vector which contains n random variables as components or elements. In the NSG, when error is represented as a random vector, its corresponding statistics are specified by a statistical error model. The corresponding random vector is also sometimes termed a random error vector.

- The realization of a Random Vector corresponds to a specific value of the vector (components or elements) for a given event such as a trial or experiment. Important descriptive statistics of a RV are its mean (vector) value and the error covariance matrix about the mean, and optionally, a multi-variate probability density function. These statistics can be predictive or sample-based.

Realization - For NSG accuracy and predicted accuracy applications, a specific trial or experimental outcome or independent sample involving a random error (category of error).

Relative Horizontal Accuracy - The range of values for the error in the difference between two objects' horizontal metric geolocation values with respect to a specified geodetic horizontal reference datum; e.g. expressed as a radial error at the 90 percent probability level (CE90). There are two types of relative horizontal accuracy: predicted relative horizontal accuracy is based on error propagation via a statistical error model(s); and measured relative horizontal accuracy is an empirically derived metric based on sample statistics.

Relative Vertical Accuracy - The range of values for the error in the difference between two objects' vertical metric geolocation values with respect to a specified geodetic vertical reference datum; e.g. expressed as a linear error at the 90 percent probability level (LE90). There are two types of relative vertical accuracy: predicted relative vertical accuracy is based on error propagation via a statistical error model(s); and measured relative vertical accuracy is an empirically derived metric based on sample statistics.

Rigorous Error Propagation - Represents the proper statistical modeling of all significant errors and their interrelationships throughout an NSG system. It enables optimal solutions as well as reliable predicted accuracies associated with specific estimates and products across the system modules.

Scalar Accuracy Metrics (augmented definition) - convenient one-number summaries of geolocation accuracy and geolocation predicted accuracy expressed as a probability: [b],[f], and [h]

- Linear Error (LE) - LE is an unsigned value that corresponds to the length of a vertical line (segment) such that there is a 90% probability that the absolute value of vertical error resides along the line. If the line is doubled in length and centered at the target solution, there is a 90% probability that the true target vertical location resides along the line. LE_XX corresponds to LE at the XX % probability level.
- Circular Error (CE) - CE is an unsigned value that corresponds to the radius of a circle such that there is a 90% probability that the horizontal error resides within the circle; or equivalently, if the circle is centered at the target solution, there is a 90% probability the true target horizontal location resides within the circle. CE_XX corresponds to CE at the XX % probability level.
- Spherical Error (SE) - SE is an unsigned value that corresponds to the radius of a sphere such that there is a 90% probability that 3d error resides within, or equivalently, if the sphere is centered at the target solution, there is a 90% probability that the true target location resides within the sphere. SE_XX corresponds to SE at the XX % probability level.

For the above scalar accuracy metrics:

- It is assumed that the underlying x-y-z coordinate system is a local tangent plane system, i.e., x and y are horizontal components and z the vertical component.
- CE-LE corresponds to the CE-LE error cylinder. There is a probability between 81 to 90 percent that 3d radial error resides within the cylinder. The former value corresponds to uncorrelated horizontal and vertical errors, the latter value to highly correlated horizontal and vertical errors.
- LE_XX, CE_XX, and SE_XX (aka LEXX, CEXX, and SEXX, respectively) are also called XX percentiles for absolute vertical errors, horizontal radial errors, and spherical radial errors, respectively. XX is expressed as an integer greater than zero and less than 100.

Standard Deviation – The square root of the variance of a random variable. A measure of deviation or dispersion about the random variable's mean-value.

State Vector - A vector of parameters or variables that describe a system's state.

State Vector Error - A vector of errors corresponding to an estimate of a state vector relative to a (typically unknown) true state vector; a random vector of errors, or random error vector.

Uncertainty – A lack of certainty; limited knowledge; unknown or imperfect information. In terms of NSG applications, more general than the concepts of errors and accuracy, but sometimes used informally as a synonym. Applies to predicted accuracy but not to empirical (sample-based) accuracy.

Uncorrelated Error - At an intuitive level, an error that is statically unrelated to all other relevant errors. More precisely, if two random variables represent two uncorrelated errors (about their respective mean-values), their covariance and their corresponding correlation coefficient are zero. A random variable is uncorrelated (with itself) from one realization to the next by definition. This latter property is also true for the random variables making up random vectors, stochastic processes, and random fields. However, these three representations typically include correlated errors within the same realization.

Uncorrelated Values - Values (of random variables or errors) which are statistically unrelated. [f] This is represented for two random variables by their covariance with a value of zero.

Variance - The measure of the dispersion of a random variable about its mean-value, also the standard deviation squared. [b]

Vertical Error - As applied to geospatial measurements and processes, vertical error is a signed and one dimensional (linear) error value typically observed in the direction of the z-axis of a local right-handed coordinate system where the x, y plane is defined as tangent to the defined reference surface at the point of origin and the z-axis is normal to the x, y plane and positive in the up direction.

Appendix B: Accuracy Validation Pseudo-code

This Appendix contains pseudo-code to perform Validation of Accuracy Requirements. It performs the entire validation process per Sections 4.1/5.1 given the appropriate inputs as documented below.

Note that although the pseudo-code (MATLAB) may appear somewhat complicated/detailed, the underlying algorithm is not. The majority of the code involves accommodating optional inputs, error checking and warnings associated with the inputs, and formatting the output which includes plots. The code is literally a "drop in" to a MATLAB capability. Also, the pseudo-code makes use of another pseudo-code function "OrderedStatisticsBestEstimate" which computes the least-upper-bound and best estimate of the appropriate percentile using order statistics. This function is documented in Appendix C of TGD 2b.

Although they can be overridden by optional inputs, important defaults values used in the following pseudo-code are 90% for the level of the (radial error random variable) percentile, 90% for the confidence–level of the percentile's least-upper-bound, and the use of a one-sided confidence interval to compute the least-upper-bound. Also, recall that the radial error percentile can correspond to either vertical, horizontal, or (3d) radial error. The corresponding percentile is equivalent to LE_XX, CE_XX, and SE_XX, respectively, at the XX probability or percentile-level.

In addition, following the pseudo-code are examples of function calls with corresponding inputs and computed outputs.

B.1 Pseudo-Code

```
function AccuracyValidation(errVal,varargin)

%%%%% The "AccuracyValidation" function validates the accuracy of a set of
%%%%% error samples or values (errVal) supplied by the user.
%%%%%
%%%%% This function is described in the document "Accuracy and Predicted
%%%%% Accuracy in the NSG: Specification and Validation; Technical Guidance
%%%%% Document (TGD) 2c".
%%%%%
%%%%% Inputs: errVal = the error samples or values supplied by the user.
%%%%%                  This variable is a matrix consisting of n (number of
%%%%%                  samples) rows and 1, 2, or 3 columns resulting in
%%%%%                  the validation of the vertical, horizontal (radial),
%%%%%                  or 3d radial error, respectively.  This input is
%%%%%                  required and is always the first input into this
%%%%%                  function.  If a three column matrix is supplied all
%%%%%                  three (vertical, horizontal (radial), and 3d radial)
%%%%%                  accuracy validation calculations will be performed.
%%%%%
```

```
%%%%%          varargin = this a variable number of inputs that could
%%%%%                     entered into this function.  The list of possible
%%%%%                     inputs follows.  None of these variables are
%%%%%                     required as inputs because the funtion has default
%%%%%                     values set.  After each possible input definition
%%%%%                     below will be how the input should be called for
%%%%%                     use.
%%%%%
%%%%%                     percentile (probability) level (prob) = the
%%%%%                        percentile value corresponding to x sub p.  The
%%%%%                        percentile has three possible values: 50, 90,
%%%%%                        or 95 percent.  90 percent is the default
%%%%%                        value that will be used if the percentile is
%%%%%                        not set or is not set to one of the three
%%%%%                        values above.  To set the percentile value the
%%%%%                        user would enter "'Percentile',90".
%%%%%
%%%%%                     confidence level (confid) = the confidence level
%%%%%                        that applies to either the one-sided or
%%%%%                        two-sided confidence interval.  The confidence
%%%%%                        has three possible values: 50, 90, or 95
%%%%%                        percent.  90 percent is the default value that
%%%%%                        will be used if the confidence value is not
%%%%%                        set or is not set to one of the three values
%%%%%                        above.  To set the confidence value the user
%%%%%                        would enter "'Confidence',90".
%%%%%
%%%%%                     type of lub (lubType) = specifies if the one-sided
%%%%%                        or two-sided lub calculation will be performed.
%%%%%                        One-sided is the default type that will be used
%%%%%                        if the type is not set or is not set 1 or 2.
%%%%%                        To set the lub type the user would enter
%%%%%                        "'lub',1".
%%%%%
%%%%%                     spec for percentile level (XXspec) = value that
%%%%%                        the least-upper-bound (lub) must be less than
%%%%%                        for validation.  If this value is not entered
%%%%%                        this function cannot determine if the error
%%%%%                        samples pass validation.  The spec level can be
%%%%%                        set for the LE (vertical), CE (horizontal
%%%%%                        radial), and SE (3d radial) test values.  All
%%%%%                        values can be set and only the value
%%%%%                        corresponding to the test being performed will
%%%%%                        be used.  To set the spec level the user would
%%%%%                        enter "'LEspec',5.5".
%%%%%
%%%%%                     error bound for radial errors (XXmax) = this
%%%%%                        value is used to pervent very large outliers.
%%%%%                        The error bound level can be set for the LE max
%%%%%                        (vertical), CE max (horizontal radial), and SE
%%%%%                        max (3d radial) test values.  All values can be
%%%%%                        set and only the value corresponding to the
%%%%%                        test being performed will be used.  To set the
%%%%%                        error bound level the user would enter
%%%%%                        "'LEmax',6".
%%%%%
```

```
%%%%%                    minimum number of i.i.d. error samples (minSamp) =
%%%%%                       minimum number of independent sample values for
%%%%%                       testing validation as specified for the current
%%%%%                       system/process being validated.  To set the
%%%%%                       minimum number of i.i.d. error samples value
%%%%%                       the user would enter "'MinSamples',50".
%%%%%
%%%%% Outputs: Statements of the results are printed to the Command Window.
%%%%%          The use of the Ordered Statistics function as found in
%%%%%          "Accuracy and Predicted Accuracy in the NSG: Sample
%%%%%          Statistics; Technical Guidance Document TGD 2b Section 4:
%%%%%          Order Statistics Appendix C" was used to obtain the values
%%%%%          printed in those statements and the values are:
%%%%%          The best estimate position and value for each validation
%%%%%          test corresponds to the the k** and y sub k** values.
%%%%%          The least-upper-bound (lub) position and value correspond to
%%%%%          the k^ and y sub k^ for the one-sided test.  For the
%%%%%          two-sided test the values correspond to k+r and y sub k+r.
%%%%%
%%%%%          Figures of sample error values, least-upper-bound, and the
%%%%%          best estimate.

%%%%% CHECKING INPUTS
%%% Checks that the error sample value matrix is oriented correctly.
errSize = size(errVal);              % checks size of error value matrix
if errSize(1) < errSize(2)           % compares number of rows to columns
    errVal  = transpose(errVal);     % transposes error value matrix
end                                  % ends loop comparing rows and columns
if size(errVal,2) > 3                % checks number of error components
    fprintf('\n\nThis Accuracy Validation function does not handle data ')
    fprintf('sets larger than 3d.\n\n') % prints statement to command window
    return                           % exits accuracy validation function
end                                  % ends loop checking error components
if size(errVal,1) < 50               % checks number of error sample values
    fprintf('\n\nWarning: The number of Error Samples entered ')
    fprintf('(%i) is smaller than the recommended number ',size(errVal,1))
    fprintf('of 50.\n')              % prints statement to command window
end                                  % ends loop checking number of values

numSamp = size(errVal,1);            % number of error samples being validated
fprintf('\n\nThe current Accuracy Validation activity is validating ')
fprintf('%i samples.\n',numSamp)     % prints statement to command window
testFlag    = size(errVal,2);        % flag for validation tests to perform

%%% Checking optional input items.
prob    = 90;                        % default value for percentile level
confid  = 90;                        % default value for confidence level
lubType = 1;                         % default value for lub type
LEspec  = [];                        % creates variable for LE spec value
CEspec  = [];                        % creates variable for CE spec value
SEspec  = [];                        % creates variable for SE spec value
LEmax   = [];                        % creates variable for LE spec max value
CEmax   = [];                        % creates variable for CE spec max value
SEmax   = [];                        % creates variable for SE spec max value
minSamp = [];                        % creates variable for min num samples
```

```
if nargin > 1                          % checks optional inputs were entered
    for n = 1:2:nargin-2               % starts loop to cycle through inputs
        if strcmp(varargin{n},'Percentile') % checks if setting percentile
            if isnumeric(varargin{n+1}) % checks entered value is a number
                if varargin{n+1} == 50 ||...
                   varargin{n+1} == 90 ||...
                   varargin{n+1} == 95  % checks entered value is option
                    prob    = varargin{n+1};    % updates percentile value
                end                    % ends loop checking entered options
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'Confidence') %checks if setting confidence
            if isnumeric(varargin{n+1}) % checks entered value is a number
                if varargin{n+1} == 50 ||...
                   varargin{n+1} == 90 ||...
                   varargin{n+1} == 95  % checks entered value is option
                    confid  = varargin{n+1};    % updates confidence value
                end                    % ends loop checking entered options
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'lub')    % checks if lub type is being set
            if isnumeric(varargin{n+1})     % checks entered value is number
                if varargin{n+1} == 1 ||...
                   varargin{n+1} == 2   % checks entered value is option
                    lubType = varargin{n+1};    % updates lub type
                end                    % ends loop checking entered options
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'LEspec') % checks if LE spec is being set
            if isnumeric(varargin{n+1})     % checks entered value is number
                LEspec  = varargin{n+1};    % LE spec value
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'CEspec') % checks if CE spec is being set
            if isnumeric(varargin{n+1}) % checks entered value is a number
                CEspec  = varargin{n+1};    % CE spec value
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'SEspec') % checks if SE spec is being set
            if isnumeric(varargin{n+1})     % checks entered value is number
                SEspec  = varargin{n+1};    % SE spec value
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'LEmax')  % checks LE max error bound
            if isnumeric(varargin{n+1})     % checks entered value is number
                LEmax   = varargin{n+1};    % LE max error bound value
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'CEmax')  % checks CE max error bound
            if isnumeric(varargin{n+1})     % checks entered value is number
                CEmax   = varargin{n+1};    % CE max error bound value
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'SEmax')  % checks SE max error bound
            if isnumeric(varargin{n+1})     % checks entered value is number
                SEmax   = varargin{n+1};    % SE max error bound value
            end                        % ends loop checking entered value
        elseif strcmp(varargin{n},'MinSamples') %checks min number of samples
            if isnumeric(varargin{n+1})     % checks entered value is number
                minSamp = varargin{n+1};    % min number of error samples
            end                        % ends loop checking entered value
        end                            % ends loop checking provided inputs
    end                                % ends loop cycling through inputs
end                                    % ends loop checking for optional inputs
```

```
fprintf('The Percentile (Probability) Level being used ')
fprintf('is %i%%',prob)              % prints statement to command window
if prob == 90                        % checks probability level being used
    fprintf(' (default).\n')         % prints statement to command window
else                                 % checks probability level being used
    fprintf('.\n')                   % prints statement to command window
end                                  % ends loop checking probability level
fprintf('The Confidence Level being used ')
fprintf('is %i%%',confid)            % prints statement to command window
if confid == 90                      % checks confidence level being used
    fprintf(' (default).\n')         % prints statement to command window
else                                 % checks confidence level being used
    fprintf('.\n')                   % prints statement to command window
end                                  % ends loop checking confidence level
fprintf('The lub (least-upper-bound) type being used ')
fprintf('is %i-sided',lubType)       % prints statement to command window
if lubType == 1                      % checks lub type being used
    fprintf(' (default).\n')         % prints statement to command window
else                                 % checks lub type being used
    fprintf('.\n')                   % prints statement to command window
end                                  % ends loop checking lub type
if isempty(minSamp) == 0             % checks min number of error samples
    if numSamp < minSamp             % checks if number of samples is met
        fprintf('WARNING: The number of error samples (%i) ',numSamp)
        fprintf('is less than the minimum number of i.i.d. error ')
        fprintf('samples (%i).\n',minSamp)  % prints statement
    end                              % ends loop checking number of samples
end                                  % ends loop checking min number of i.i.d.

%%%%% CHECKING INPUTS

%%% Use of the Ordered Statistics function as found in "Accuracy and
%%% Predicted Accuracy in the NSG: Sample Statistics; Technical Guidance
%%% Document TGD 2b Section 4: Order Statistics Appendix C".  Example
%%% function call:
%%% [bestEst,oneSided,twoSided,y] =
%%%                     OrderedStatisticsBestEstimate(prob,confid,verErr);

while testFlag == 1 || testFlag == 3    % checks if validating vertical
    if testFlag == 1                 % checks if validating only vertical
        fprintf('\n\nThe vertical component of the sample errors will ')
        fprintf('be analyzed (validated).\n\n') % prints statement
        verErr  = abs(errVal);       % vertical error value
    else                             % checks if validating all 3 components
        fprintf('\n\nThe vertical, horizontal (radial), and 3d radial ')
        fprintf('errors will be analyzed (validated).\n\n') %prints statement
        verErr  = abs(errVal(:,3)); % vertical error values
    end                              % ends loop checking number of components

    fprintf('\n***NOTE: The following results for the vertical error ')
    fprintf('samples are from the Ordered Statistics function found in ')
    fprintf('Appendix C of TGD 2b.***\nThese results contain detailed ')
    fprintf('information and the user can go directly to the final ')
    fprintf('validation results below if so desired.\n') % prints statement
    [bestEst,oneSided,twoSided,y] = ... % calls Ordered Statistics function
        OrderedStatisticsBestEstimate(prob,confid,verErr);
```

```
%%% Printing ordered sample number under ordered sample values
for m = 1:numSamp                % starts loop to cycle through samples
    for n = 1:length(sprintf('%.2f',y(m)))-...
              length(sprintf('%i',m)) % starts loop to print spaces
        fprintf(' ')             % prints spaces to command window
    end                          % ends loop printing spaces
    fprintf('%i\t\t',m)          % prints sample number
end                              % ends loop cycling thorugh samples
fprintf('\n')                    % prints statement to command window
fprintf('***NOTE: This concludes the results from the Ordered ')
fprintf('Statistics function found in Appendix C of TGD 2b.***\n')

%%% Prints validation results
fprintf('\n\nFINAL VALIDATION RESULTS: The following results are for ')
fprintf('Vertical Accuracy Validation.\n') % prints statement
verBestEst  = bestEst(end);      % vertical error best estimate value
verBestEstp = bestEst(end-1);    % vertical error best estimate position
if lubType == 1                  % checks lub type being used
    if oneSided(2) == 0          % checks if lub was found
        fprintf('WARNING: Not enough samples to compute lub for ')
        fprintf('vertical validation.\n')   % prints statement
        fprintf('Computations have been stopped for the ')
        fprintf('vertical analysis.\n') % prints statement
        break                    % breaks from while loop
    elseif oneSided(2) == numSamp   % checks if lub is last
        fprintf('WARNING: The lub value was computed using the ')
        fprintf('last ordered sample.\n')   % prints statement
    end                          % ends loop checking lub value
    verLUB  = oneSided(3);       % vertical least-upper-bound
    verLUBp = oneSided(2);       % vertical least-upper-bound position
else                             % action when two-sided lub being used
    if twoSided(4) == 0          % checks if lub was found
        fprintf('WARNING: Not enough samples to compute lub value ')
        fprintf('for vertical validation.\n')   % prints statement
        fprintf('Computations have been stopped for the ')
        fprintf('vertical analysis.\n') % prints statement
        break                    % breaks from while loop
    elseif twoSided(4) == numSamp   % checks if lub is last
        fprintf('WARNING: The lub value for vertical validation ')
        fprintf('was computed using the last ordered sample.\n')
    end                          % ends loop checking lub value
    verLUB  = twoSided(5);       % vertical least-upper-bound
    verLUBp = twoSided(4);       % vertical least-upper-bound position
end                              % ends loop checking lub type

if isempty(LEspec) && isempty(LEmax) % checks spec values
    fprintf('No spec value or max spec value were supplied for ')
    fprintf('vertical validation.\n') % prints statement
elseif isempty(LEspec) == 0 && isempty(LEmax) % checks spec values
    if verLUB < LEspec           % spec value test
        fprintf('The vertical error samples PASS validation for the ')
        fprintf('spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',verLUB)
        fprintf('Spec Value: %.2f\n',LEspec) % prints statement
        fprintf('No max spec value was supplied for vertical ')
        fprintf('validation.\n') % prints statement to command window
```

```
    else                        % spec value test
        fprintf('The vertical error samples FAIL validation for the ')
        fprintf('spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',verLUB)
        fprintf('Spec Value: %.2f\n',LEspec) % prints statement
        fprintf('No max spec value was supplied for vertical ')
        fprintf('validation.\n') % prints statement to command window
    end                         % ends loop for spec value test
elseif isempty(LEspec) && isempty(LEmax) == 0 % checks spec values
    if max(verErr) < LEmax      % max spec value test
        fprintf('The vertical error samples PASS validation for the ')
        fprintf('max spec value test.\n') % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(verErr))
        fprintf('Max Spec Value: %.2f\n',LEmax) % prints statement
        fprintf('No spec value was supplied for vertical ')
        fprintf('validation.\n') % prints statement to command window
    else                        % max spec value test
        fprintf('The vertical error samples FAIL validation for the ')
        fprintf('max spec value test.\n') % prints statement
        loc = find(verErr > LEmax); % values greater than LE max spec
        fprintf('\tVertical error ') % prints statement to command window
        if length(loc) == 1     % checks number of locations found
            fprintf('sample %i ',loc)
            fprintf('(%.2f) is ',verErr(loc))   % prints statement
        else                    % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
                fprintf('%i (%.2f) ',loc(1),verErr(loc(1)))
                fprintf('and %i (%.2f) are ',loc(2),verErr(loc(2)))
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',verErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',verErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations


        fprintf('greater than the LE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(verErr))    % prints statement
        fprintf('No spec value was supplied for vertical validation.\n')
    end                         % ends loop for max spec value test
else                            % checks spec values
    if verLUB < LEspec &&...
       max(verErr) < LEmax      % checks that vertical passes validation
        fprintf('The vertical error samples PASS validation for the ')
        fprintf('spec value test and the max spec value test.\n')
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',verLUB)
        fprintf('Spec Value: %.2f\n',LEspec) % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(verErr))
        fprintf('Max Spec Value: %.2f\n',LEmax) % prints statement
```

```
elseif verLUB > LEspec &&...
       max(verErr) > LEmax  % action when vertical fails validation
    fprintf('The vertical error samples FAIL validation due to ')
    fprintf('the spec value test and max spec value test.\n')
    fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',verLUB)
    fprintf('Spec Value: %.2f\n',LEspec) % prints statement
    loc = find(verErr > LEmax); % values greater than LE max spec
    fprintf('\tVertical error ') % prints statement to command window
    if length(loc) == 1     % checks number of locations found
        fprintf('sample %i ',loc)
        fprintf('(%.2f) is ',verErr(loc))   % prints statement
    else                    % action when multiple locations found
        fprintf('samples ') % prints statement to command window
        if length(loc) == 2 % checks length of location vector
            fprintf('%i (%.2f) ',loc(1),verErr(loc(1)))
            fprintf('and %i (%.2f) are ',loc(2),verErr(loc(2)))
        else                % checks length of location vector
            for n = 1:length(loc)   % starts loop to cycle locations
                if n == length(loc) % checks current location
                    fprintf('and %i ',loc(n))   % prints statement
                    fprintf('(%.2f) are ',verErr(loc(n)))
                else        % checks current location position
                    fprintf('%i ',loc(n))       % prints statement
                    fprintf('(%.2f), ',verErr(loc(n)))
                end         % ends checking current location position
            end             % ends loop cycling thorugh locations
        end                 % ends loop checking length of loc vector
    end                     % ends loop checking number of locations
    fprintf('greater than the LE max spec value (%i of ',length(loc))
    fprintf('%i samples).\n',length(verErr))    % prints statement
elseif verLUB > LEspec      % checks validation test
    fprintf('The vertical error samples FAIL validation due to ')
    fprintf('the spec value test.\n')   % prints statement
    fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',verLUB)
    fprintf('Spec Value: %.2f\n',LEspec) % prints statement
    fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(verErr))
    fprintf('Max Spec Value: %.2f\n',LEmax) % prints statement


elseif max(verErr) > LEmax  % checks validation test
    fprintf('The vertical error samples FAIL validation due to ')
    fprintf('the max spec value test.\n')   % prints statement
    fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(verErr))
    fprintf('Max Spec Value: %.2f\n',LEmax) % prints statement
    loc = find(verErr > LEmax); % values greater than LE max spec
    fprintf('\tVertical error ') % prints statement to command window
    if length(loc) == 1     % checks number of locations found
        fprintf('sample %i ',loc)
        fprintf('(%.2f) is ',verErr(loc))   % prints statement
    else                    % action when multiple locations found
        fprintf('samples ') % prints statement to command window
        if length(loc) == 2 % checks length of location vector
            fprintf('%i (%.2f) ',loc(1),verErr(loc(1)))
            fprintf('and %i (%.2f) are ',loc(2),verErr(loc(2)))
```

```
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',verErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',verErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations
        fprintf('greater than the LE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(verErr))    % prints statement
    end                         % ends loop checking vertical validation
end                             % ends loop checking spec values

fprintf('The lub value occurs at ordered sample %i of ',verLUBp)
fprintf('%i, corresponding to a value of %.2f\n',numSamp,verLUB)
fprintf('The Best Estimate of the percentile is at ')
fprintf('ordered sample ')      % prints statement to command window
if mod(verBestEstp,1) == 0      % checks if best estimate position
    fprintf('%i of %i, corresponding to a ',verBestEstp,numSamp)
    fprintf('value of %.2f\n',verBestEst)   % prints statment
else                            % checks if best estimate position
    fprintf('%.1f of %i, corresponding to a ',verBestEstp,numSamp)
    fprintf('value of %.2f\n',verBestEst)   % prints statment
end                             % ends loop checking best estimate
fprintf('\n\n')                 % prints statement to command window

%%% Plotting vertical values
legFlag = 1;                    % creates legend flag for current figure
figure(1)                       % makes figure current
clf                             % clears current figure
hold on                         % turns hold on for current figure
plot(1:numSamp,verErr,'b.')     % plots vertical error values
plot([0 numSamp],verLUB*[1 1],'m-') % plots vertical least-upper-bound
plot([0 numSamp],verBestEst*[1 1],'r--')    % plots best estimate value


if isempty(LEspec) == 0         % checks LE spec value was supplied
    plot([0 numSamp],LEspec*[1 1],'k-') % plots LE spec value
    legFlag = 2;                % updates legend flag for current figure
end                             % ends loop checking LE spec value
title(['Vertical Error Sample (Probability Level: ' num2str(prob)...
       '%;Confidence Level: ' num2str(confid) '%)'])    % adds title
xlabel('Sample Number')         % adds label to x-axis
ylabel('Error Value')           % adds label to y-axis
if legFlag == 1                 % checks legend flag value
    legend('Sample Value','Least-upper-bound',...
           'Best Estimate')     % adds legend to figure
```

```
    elseif legFlag == 2             % checks legend flag value
        legend('Sample Value','Least-upper-bound',...
               'Best Estimate','Spec Value')    % adds legend to figure
    end                             % ends loop checking legend flag
    hold off                        % turns hold off for current figure
    drawnow                         % forces display to update
    break                           % breaks from while loop
end                                 % ends loop validating vertical

while testFlag == 2 || testFlag == 3 % checks if validating horizontal
    if testFlag == 2                % checks if validating only horizontal
        fprintf('\n\nThe horizontal (radial) component of the sample ')
        fprintf('errors will be analyzed (validated).\n') % prints statement
    end                             % ends loop checking number of components

    fprintf('\n***NOTE: The following results for the horizontal (radial) ')
    fprintf('error samples are from the Ordered Statistics function found ')
    fprintf('in Appendix C of TGD 2b.***\nThese results contain detailed ')
    fprintf('information and the user can go directly to the final ')
    fprintf('validation results below if so desired.\n') % prints statement
    horErr  = sqrt(errVal(:,1).^2+errVal(:,2).^2);  % horizontal error value
    [bestEst,oneSided,twoSided,y] = ... % calls Ordered Statistics function
                    OrderedStatisticsBestEstimate(prob,confid,horErr);
    %%% Printing ordered sample number under ordered sample values
    for m = 1:numSamp               % starts loop to cycle through samples
        for n = 1:length(sprintf('%.2f',y(m)))-...
                length(sprintf('%i',m)) % starts loop to print spaces
            fprintf(' ')            % prints spaces to command window
        end                         % ends loop printing spaces
        fprintf('%i\t\t',m)         % prints sample number
    end                             % ends loop cycling thorugh samples
    fprintf('\n')                   % prints statement to command window
    fprintf('***NOTE: This concludes the results from the Ordered ')
    fprintf('Statistics function found in Appendix C of TGD 2b.***\n')

    %%% Prints validation results
    fprintf('\n\nFINAL VALIDATION RESULTS: The following results are for ')
    fprintf('Horizontal (radial) Accuracy Validation.\n') % prints statement
    horBestEst  = bestEst(end);     % horizontal radial best estimate value
    horBestEstp = bestEst(end-1);   % horizontal best estimate position


    if lubType == 1                 % checks lub type being used
        if oneSided(2) == 0         % checks if lub was found
            fprintf('WARNING: Not enough samples to compute lub value ')
            fprintf('for horizontal (radial) validation.\n')
            fprintf('Computations have been stopped for the horizontal ')
            fprintf('(radial) analysis.\n') % prints statement
            break                   % breaks from while loop
        elseif oneSided(2) == numSamp   % checks if lub is last
            fprintf('WARNING: The lub value for horizontal (radial) ')
            fprintf('validation was computed using the last ordered ')
            fprintf('sample.\n')    % prints statement to command window
        end                         % ends loop checking lub value
        horLUB  = oneSided(3);      % horizontal radial least-upper-bound
        horLUBp = oneSided(2);      % horizontal least-upper-bound position
```

```
else                            % action when two-sided lub being used
    if twoSided(4) == 0         % checks if lub was found
        fprintf('WARNING: Not enough samples to compute lub value ')
        fprintf('for horizontal (radial) validation.\n')
        fprintf('Computations have been stopped for the horizontal ')
        fprintf('(radial) analysis.\n') % prints statement
        break                   % breaks from while loop
    elseif twoSided(4) == numSamp   % checks if lub is last
        fprintf('WARNING: The lub value for horizontal (radial) ')
        fprintf('validation was computed using the last ordered ')
        fprintf('sample.\n')    % prints statement to command window
    end                         % ends loop checking lub value
    horLUB  = twoSided(5);      % horizontal radial least-upper-bound
    horLUBp = twoSided(4);      % horizontal least-upper-bound position
end                             % ends loop checking lub type

if isempty(CEspec) && isempty(CEmax) % checks spec values
    fprintf('No spec value or max spec value were supplied for ')
    fprintf('horizontal (radial) validation.\n') % prints statement
elseif isempty(CEspec) == 0 && isempty(CEmax) % checks spec values
    if horLUB < CEspec          % spec value test
        fprintf('The horizontal (radial) error samples PASS validation ')
        fprintf('for the spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',horLUB)
        fprintf('Spec Value: %.2f\n',CEspec)
        fprintf('No max spec value was supplied for horizontal ')
        fprintf('(radial) validation.\n') % prints statement
    else                        % spec value test
        fprintf('The horizontal (radial) error samples FAIL validation ')
        fprintf('for the spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',horLUB)
        fprintf('Spec Value: %.2f\n',CEspec) % prints statement
        fprintf('No max spec value was supplied for horizontal ')
        fprintf('(radial) validation.\n') % prints statement
    end                         % ends loop checking spec value test


elseif isempty(CEspec) && isempty(CEmax) == 0 % checks spec values
    if max(horErr) < CEmax      % max spec value test
        fprintf('The horizontal (radial) error samples PASS validation ')
        fprintf('for the max spec value test.\n') % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(horErr))
        fprintf('Max Spec Value: %.2f\n',CEmax) % prints statement
        fprintf('No spec value was supplied for horizontal (radial) ')
        fprintf('validation.\n') % prints statement to command window
    else                        % max spec value
        fprintf('The horizontal (radial) error samples FAIL validation ')
        fprintf('for the max spec value test.\n') % prints statement
        loc = find(horErr > CEmax); % values greater than CE max spec
        fprintf('\tHorizontal (radial) error ') % prints statement
        if length(loc) == 1     % checks number of locations found
            fprintf('sample %i ',loc)
            fprintf('(%.2f) is ',horErr(loc))   % prints statement
        else                    % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
```

```
                fprintf('%i (%.2f) ',loc(1),horErr(loc(1)))
                fprintf('and %i (%.2f) are ',loc(2),horErr(loc(2)))
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',horErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',horErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations
        fprintf('greater than the CE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(horErr))    % prints statement
        fprintf('No spec value was supplied for horizontal (radial) ')
        fprintf('validation.\n') % prints statement to command window
    end                         % ends loop checking max spec value test
else                            % checks spec values
    if horLUB < CEspec &&...
       max(horErr) < CEmax     % checks that horizontal passes validation
        fprintf('The horizontal (radial) error samples PASS validation ')
        fprintf('for the spec value test and the max spec value test.\n')
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',horLUB)
        fprintf('Spec Value: %.2f\n',CEspec)
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(horErr))
        fprintf('Max Spec Value: %.2f\n',CEmax) % prints statement
    elseif horLUB > CEspec &&...
           max(horErr) > CEmax  % action when horizontal fails validation
        fprintf('The horizontal (radial) error samples FAIL validation ')
        fprintf('due to the spec value test and max spec value test.\n')
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',horLUB)
        fprintf('Spec Value: %.2f\n',CEspec) % prints statement
        loc = find(horErr > CEmax); % values greater than CE max spec
        fprintf('\tHorizontal (radial) error ') % prints statement


        if length(loc) == 1     % checks number of locations found
            fprintf('sample %i ',loc)
            fprintf('(%.2f) is ',horErr(loc))   % prints statement
        else                    % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
                fprintf('%i (%.2f) ',loc(1),horErr(loc(1)))
                fprintf('and %i (%.2f) are ',loc(2),horErr(loc(2)))
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',horErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',horErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
```

```
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations
        fprintf('greater than the CE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(horErr))    % prints statement
    elseif horLUB > CEspec      % checks validation test
        fprintf('The horizontal (radial) error samples FAIL validation ')
        fprintf('due to the spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',horLUB)
        fprintf('Spec Value: %.2f\n',CEspec)
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(horErr))
        fprintf('Max Spec Value: %.2f\n',CEmax)
    elseif max(horErr) > CEmax  % checks validation test
        fprintf('The horizontal (radial) error samples FAIL validation ')
        fprintf('due to the max spec value test.\n') % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(horErr))
        fprintf('Max Spec Value: %.2f\n',CEmax)
        loc = find(horErr > CEmax); % values greater than CE max spec
        fprintf('\tHorizontal (radial) error ') % prints statement
        if length(loc) == 1     % checks number of locations found
            fprintf('sample %i ',loc)
            fprintf('(%.2f) is ',horErr(loc))   % prints statement
        else                    % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
                fprintf('%i (%.2f) ',loc(1),horErr(loc(1)))
                fprintf('and %i (%.2f) are ',loc(2),horErr(loc(2)))
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',horErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',horErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations
        fprintf('greater than the CE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(horErr))    % prints statement
    end                         % ends loop checking vertical validation
end                             % ends loop checking spec values

fprintf('The lub value occurs at ordered sample %i of ',horLUBp)
fprintf('%i, corresponding to a value of %.2f\n',numSamp,horLUB)
fprintf('The Best Estimate of the percentile is at ordered sample ')
if mod(horBestEstp,1) == 0      % checks best estimate position
    fprintf('%i of %i, corresponding to a value ',horBestEstp,numSamp)
    fprintf('of %.2f\n',horBestEst) % prints statment to command window
else                            % checks best estimate position
    fprintf('%.1f of %i, corresponding to a value ',horBestEstp,numSamp)
    fprintf('of %.2f\n',horBestEst) % prints statment to command window
end                             % ends loop checking best estimate
fprintf('\n\n')                 % prints statement to command window
```

```
    %%% Plotting horizontal values
    legFlag = 1;                        % creates legend flag for current figure
    figure(2)                           % makes figure current
    clf                                 % clears current figure
    hold on                             % turns hold on for current figure
    plot(1:numSamp,horErr,'b.')         % plots horizontal error values
    plot([0 numSamp],horLUB*[1 1],'m-') % plots horizontal radial lub
    plot([0 numSamp],horBestEst*[1 1],'r--') % plots best estimate value
    if isempty(CEspec) == 0             % checks CE spec value was supplied
        plot([0 numSamp],CEspec*[1 1],'k-') % plots CE spec value
        legFlag = 2;                    % updates legend flag for current figure
    end                                 % ends loop checking CE spec value
    title(['Horizontal (Radial) Error Sample (Probability Level: ' ...
           num2str(prob) '%;Confidence Level: ' num2str(confid) '%)'])
    xlabel('Sample Number')             % adds label to x-axis
    ylabel('Error Value')               % adds label to y-axis
    if legFlag == 1                     % checks legend flag value
        legend('Sample Value','Least-upper-bound','Best Estimate')
    elseif legFlag == 2                 % checks legend flag value
        legend('Sample Value','Least-upper-bound','Best Estimate',...
               'Spec Value')            % adds legend to figure
    end                                 % ends loop checking legend flag
    hold off                            % turns hold off for current figure
    drawnow                             % forces display to update
    break                               % breaks from while loop
end                                     % ends loop validating horizontal

while testFlag == 3                     % checks if validating 3d radial
    fprintf('\n***NOTE: The following results for the 3d radial error ')
    fprintf('samples are from the Ordered Statistics function found in ')
    fprintf('Appendix C of TGD 2b.***\nThese results contain detailed ')
    fprintf('information and the user can go directly to the final ')
    fprintf('validation results below if so desired.\n')
    radErr  = sqrt(errVal(:,1).^2+errVal(:,2).^2+errVal(:,3).^2); % 3d errors
    [bestEst,oneSided,twoSided,y] = ... % calls Ordered Statistics function
                        OrderedStatisticsBestEstimate(prob,confid,radErr);


    %%% Printing ordered sample number under ordered sample values
    for m = 1:numSamp                   % starts loop to cycle through samples
        for n = 1:length(sprintf('%.2f',y(m)))-...
                length(sprintf('%i',m)) % starts loop to print spaces
            fprintf(' ')                % prints spaces to command window
        end                             % ends loop printing spaces
        fprintf('%i\t\t',m)             % prints sample number
    end                                 % ends loop cycling thorugh samples
    fprintf('\n')                       % prints statement to command window
    fprintf('***NOTE: This concludes the results from the Ordered ')
    fprintf('Statistics function found in Appendix C of TGD 2b.***\n')

    %%% Prints validation results
    fprintf('\n\nFINAL VALIDATION RESULTS: The following results are for ')
    fprintf('3d Radial Accuracy Validation.\n') % prints statement
    radBestEst  = bestEst(end);         % 3d radial best estimate value
    radBestEstp = bestEst(end-1);       % 3d radial best estimate position
```

```
if lubType == 1                 % checks lub type being used
    if oneSided(2) == 0         % checks if lub was found
        fprintf('WARNING: Not enough samples to compute lub value ')
        fprintf('for 3d radial validation.\n')
        fprintf('Computations have been stopped for the 3d radial ')
        fprintf('analysis.\n')  % prints statement to command window
        break                   % breaks from while loop
    elseif oneSided(2) == numSamp   % checks if lub is last
        fprintf('WARNING: The lub value for 3d radial validation ')
        fprintf('was computed using the last ordered sample.\n')
    end                         % ends loop checking lub value
    radLUB  = oneSided(3);      % 3d radial least-upper-bound
    radLUBp = oneSided(2);      % 3d radial least-upper-bound position
else                            % action when two-sided lub being used
    if twoSided(4) == 0         % checks if lub was found
        fprintf('WARNING: Not enough samples to compute lub value ')
        fprintf('for 3d radial validation.\n')
        fprintf('Computations have been stopped for the 3d radial ')
        fprintf('analysis.\n')  % prints statement to command window
        break                   % breaks from while loop
    elseif twoSided(4) == numSamp   % checks if lub is last
        fprintf('WARNING: The lub value for 3d radial validation ')
        fprintf('was computed using the last ordered sample.\n')
    end                         % ends loop checking lub value
    radLUB  = twoSided(5);      % 3d radial least-upper-bound
    radLUBp = twoSided(4);      % 3d radial least-upper-bound position
end                             % ends loop checking lub type

if isempty(SEspec) && isempty(SEmax) % checks spec values
    fprintf('No spec value or max spec value were supplied for 3d ')
    fprintf('radial validation.\n') % prints statement to command window


elseif isempty(SEspec) == 0 && isempty(SEmax) % checks spec values
    if radLUB < SEspec          % spec value test
        fprintf('The 3d radial error samples PASS validation for the ')
        fprintf('spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',radLUB)
        fprintf('Spec Value: %.2f\n',SEspec)
        fprintf('No max spec value was supplied for 3d radial ')
        fprintf('validation.\n') % prints statement to command window
    else                        % spec value test
        fprintf('The 3d radial error samples FAIL validation for the ')
        fprintf('spec value test.\n') % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',radLUB)
        fprintf('Spec Value: %.2f\n',SEspec)
        fprintf('No max spec value was supplied for 3d radial ')
        fprintf('validation.\n') % prints statement to command window
    end                         % ends loop checks spec value test
elseif isempty(SEspec) && isempty(SEmax) == 0 % checks spec values
    if max(radErr) < SEmax      % max spec value test
        fprintf('The 3d radial error samples PASS validation for the ')
        fprintf('max spec value test.\n') % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(radErr))
        fprintf('Max Spec Value: %.2f\n',SEmax) % prints statement
        fprintf('No spec value was supplied for 3d radial ')
```

```
        fprintf('validation.\n') % prints statement to command window
    else                         % max spec value test
        fprintf('The 3d radial error samples FAIL validation for the ')
        fprintf('max spec value test.\n') % prints statement
        loc = find(radErr > SEmax); % values greater than SE max spec
        fprintf('\t3d radial error ') % prints statement
        if length(loc) == 1      % checks number of locations found
            fprintf('sample %i ',loc) % prints statement
            fprintf('(%.2f) is ',radErr(loc))   % prints statement
        else                     % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
                fprintf('%i (%.2f) ',loc(1),radErr(loc(1)))
                fprintf('and %i (%.2f) are ',loc(2),radErr(loc(2)))
            else                 % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',radErr(loc(n)))
                    else         % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',radErr(loc(n)))
                    end          % ends checking current location position
                end              % ends loop cycling thorugh locations
            end                  % ends loop checking length of loc vector
        end                      % ends loop checking number of locations
        fprintf('greater than the SE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(radErr))    % prints statement
        fprintf('No spec value was supplied for 3d radial validation.\n')
    end                          % ends loop checking max spec value test


else                             % checks spec values
    if radLUB < SEspec &&...
       max(radErr) < SEmax     % checks that 3d radial passes validation
        fprintf('The 3d radial error samples PASS validation for the ')
        fprintf('spec value test and the max spec value test.\n')
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',radLUB)
        fprintf('Spec Value: %.2f\n',SEspec) % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(radErr))
        fprintf('Max Spec Value: %.2f\n',SEmax) % prints statement
    elseif radLUB > SEspec &&...
           max(radErr) > SEmax  % action when horizontal fails validation
        fprintf('The 3d radial error samples FAIL validation due to ')
        fprintf('the spec value test and max spec value test.\n')
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',radLUB)
        fprintf('Spec Value: %.2f\n',SEspec) % prints statement
        loc = find(radErr > SEmax); % values greater than SE max spec
        fprintf('\t3d radial error ') % prints statement
        if length(loc) == 1      % checks number of locations found
            fprintf('sample %i ',loc)
            fprintf('(%.2f) is ',radErr(loc))   % prints statement
        else                     % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
                fprintf('%i (%.2f) ',loc(1),radErr(loc(1)))
```

```
                fprintf('and %i (%.2f) are ',loc(2),radErr(loc(2)))
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',radErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',radErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations
        fprintf('greater than the SE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(radErr))    % prints statement
    elseif radLUB > SEspec      % checks validation test
        fprintf('The 3d radial error samples FAIL validation due to ')
        fprintf('the spec value test.\n')   % prints statement
        fprintf('\tLeast-Upper-Bound Value: %.2f\t\t',radLUB)
        fprintf('Spec Value: %.2f\n',SEspec) % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(radErr))
        fprintf('Max Spec Value: %.2f\n',SEmax) % prints statement
    elseif max(radErr) > SEmax  % checks validation test
        fprintf('The 3d radial error samples FAIL validation due to ')
        fprintf('the max spec value test.\n')   % prints statement
        fprintf('\tMax Error Sample: %.2f\t\t\t\t',max(radErr))
        fprintf('Max Spec Value: %.2f\n',SEmax) % prints statement
        loc = find(radErr > SEmax); % values greater than SE max spec
        fprintf('\t3d radial error ') % prints statement


        if length(loc) == 1     % checks number of locations found
            fprintf('sample %i ',loc)
            fprintf('(%.2f) is ',radErr(loc))   % prints statement
        else                    % action when multiple locations found
            fprintf('samples ') % prints statement to command window
            if length(loc) == 2 % checks length of location vector
                fprintf('%i (%.2f) ',loc(1),radErr(loc(1)))
                fprintf('and %i (%.2f) are ',loc(2),radErr(loc(2)))
            else                % checks length of location vector
                for n = 1:length(loc)   % starts loop to cycle locations
                    if n == length(loc) % checks current location
                        fprintf('and %i ',loc(n))   % prints statement
                        fprintf('(%.2f) are ',radErr(loc(n)))
                    else        % checks current location position
                        fprintf('%i ',loc(n))       % prints statement
                        fprintf('(%.2f), ',radErr(loc(n)))
                    end         % ends checking current location position
                end             % ends loop cycling thorugh locations
            end                 % ends loop checking length of loc vector
        end                     % ends loop checking number of locations
        fprintf('greater than the SE max spec value (%i of ',length(loc))
        fprintf('%i samples).\n',length(radErr))    % prints statement
    end                         % ends loop checking vertical validation
end                             % ends loop checking spec values
```

```
    fprintf('The lub value occurs at ordered sample %i of ',radLUBp)
    fprintf('%i, corresponding to a value of %.2f\n',numSamp,radLUB)
    fprintf('The Best Estimate of the percentile is at ordered sample ')
    if mod(radBestEstp,1) == 0      % checks best estimate position
        fprintf('%i of %i, corresponding to a value ',radBestEstp,numSamp)
        fprintf('of %.2f\n',radBestEst) % prints statment to command window
    else                            % checks best estimate position
        fprintf('%.1f of %i, corresponding to a value ',radBestEstp,numSamp)
        fprintf('of %.2f\n',radBestEst) % prints statment to command window
    end                             % ends loop checking best estimate

    %%% Plotting 3d radial values
    legFlag = 1;                    % creates legend flag for current figure
    figure(3)                       % makes figure current
    clf                             % clears current figure
    hold on                         % turns hold on for current figure
    plot(1:numSamp,radErr,'b.')     % plots 3d radial error values
    plot([0 numSamp],radLUB*[1 1],'m-') % plots 3d radial least-upper-bound
    plot([0 numSamp],radBestEst*[1 1],'r--') % plots best estimate value
    title(['3d Radial Error Sample (Probability Level: ' num2str(prob) ...
           '%;Confidence Level: ' num2str(confid) '%)']) % adds title
    if isempty(SEspec) == 0         % checks SE spec value was supplied
        plot([0 numSamp],SEspec*[1 1],'k-') % plots SE spec value
        legFlag = 2;                % updates legend flag for figure
    end                             % ends loop checking SE spec value
    xlabel('Sample Number')         % adds label to x-axis
    ylabel('Error Value')           % adds label to y-axis


    if legFlag == 1                 % checks legend flag value
        legend('Sample Value','Least-upper-bound',...
               'Best Estimate')     % adds legend to figure
    elseif legFlag == 2             % checks legend flag value
        legend('Sample Value','Least-upper-bound','Best Estimate',...
               'Spec Value')        % adds legend to figure
    end                             % ends loop checking legend flag
    hold off                        % turns hold off for current figure
    drawnow                         % forces display to update
    break                           % breaks from while loop
end                                 % ends loop validating 3d radial
```

B.2 Examples

B.2.1 Example 1: Demonstration of the Minimum Number of Function Inputs to the Accuracy Validation Function

Example 1: Create a vector of 100 indpendent normally distributed error sample values corresponding to a possible set of vertical errors. Each random error has an *a priori* mean-value of zero and a standard deviation of 1. This example will demonstrate the minimum number of function inputs to the Accuracy Validation function and the resulting outputs.

```
errVals = randn(100,1);        % creates random error values; simple Gaussian %N(0,1) distribution used for simplicity; this generation of random error values is not part of the validation pseudo-code per se.
```

Error values resulting from the above command. The values are a 100-by-1 vector, but displayed like this to take up less space on the page.

Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value
1	0.5377	26	1.0347	51	-0.8637	76	-1.4023
2	1.8339	27	0.7269	52	0.0774	77	-1.4224
3	-2.2588	28	-0.3034	53	-1.2141	78	0.4882
4	0.8622	29	0.2939	54	-1.1135	79	-0.1774
5	0.3188	30	-0.7873	55	-0.0068	80	-0.1961
6	-1.3077	31	0.8884	56	1.5326	81	1.4193
7	-0.4336	32	-1.1471	57	-0.7697	82	0.2916
8	0.3426	33	-1.0689	58	0.3714	83	0.1978
9	3.5784	34	-0.8095	59	-0.2256	84	1.5877
10	2.7694	35	-2.9443	60	1.1174	85	-0.8045
11	-1.3499	36	1.4384	61	-1.0891	86	0.6966
12	3.0349	37	0.3252	62	0.0326	87	0.8351
13	0.7254	38	-0.7549	63	0.5525	88	-0.2437
14	-0.0631	39	1.3703	64	1.1006	89	0.2157
15	0.7147	40	-1.7115	65	1.5442	90	-1.1658
16	-0.2050	41	-0.1022	66	0.0859	91	-1.1480
17	-0.1241	42	-0.2414	67	-1.4916	92	0.1049
18	1.4897	43	0.3192	68	-0.7423	93	0.7223
19	1.4090	44	0.3129	69	-1.0616	94	2.5855
20	1.4172	45	-0.8649	70	2.3505	95	-0.6669
21	0.6715	46	-0.0301	71	-0.6156	96	0.1873
22	-1.2075	47	-0.1649	72	0.7481	97	-0.0825
23	0.7172	48	0.6277	73	-0.1924	98	-1.9330
24	1.6302	49	1.0933	74	0.8886	99	-0.4390
25	0.4889	50	1.1093	75	-0.7648	100	-1.7947

```
AccuracyValidation(errVals);  % calls accuracy validation function
```

The following is the results from the above call of the Accuracy Validation function.

The current Accuracy Validation activity is validating 100 samples.
The Percentile (Probability) Level being used is 90% (default).
The Confidence Level being used is 90% (default).
The lub (least-upper-bound) type being used is 1-sided (default).

The vertical component of the sample errors will be analyzed (validated).

NOTE: The following results for the vertical error samples are from the Ordered Statistics function found in Appendix C of TGD 2b.

These results contain detailed information and the user can go directly to the final validation results below if so desired.

The number of samples is 100

The k* location value is 90 Sample Value corresponding to the k* location is 1.71

The k** location value is 90.0 Sample Value corresponding to the k** location is 1.71

The k^ location value is 95 Sample Value corresponding to the k^ location is 2.35

The k^ confidence value is 93.32%

The one-sided confidence interval color code is GREEN

The k location value is 85 Sample Value corresponding to the k location is 1.49

The k+r location value is 95 Sample Value corresponding to the k+r location is 2.35

The confidence interval range is 10.10%

Determining desired confidence interval was successful.

The actual confidence percent contained within the computed confidence interval is 90.00%

The two-sided confidence interval color code is GREEN

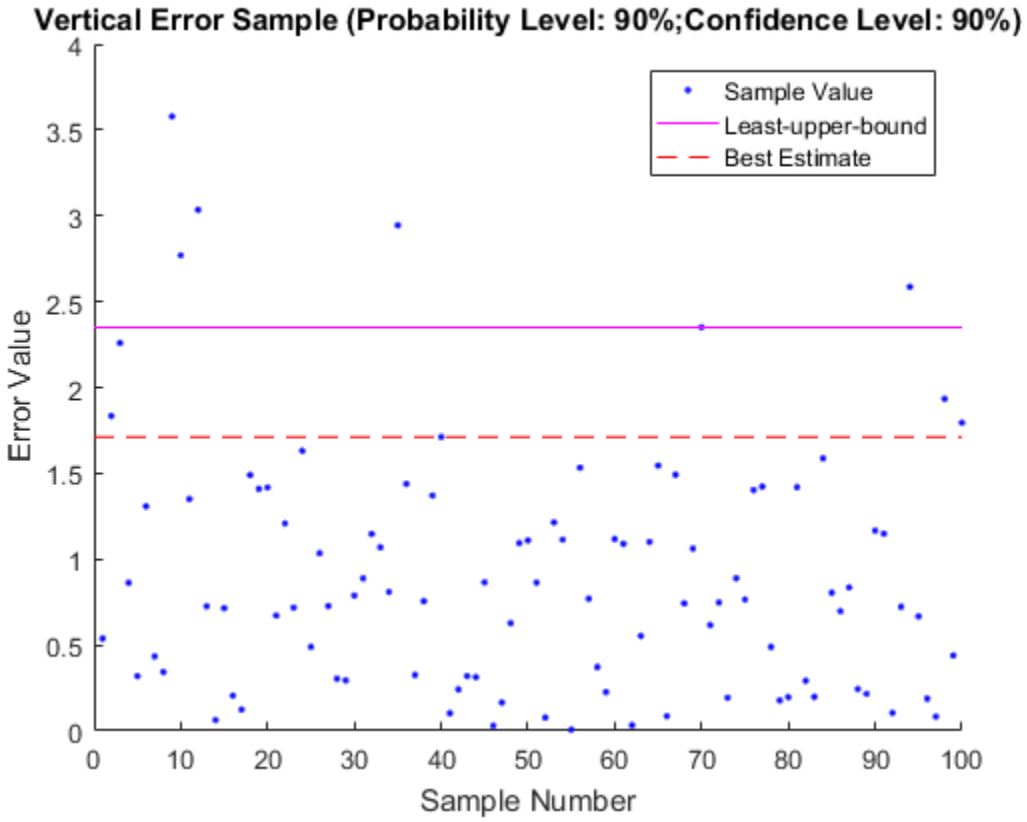
The ordered sample values are: *Again shown this way to conserve space.*

Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value
1	0.0068	26	0.3188	51	0.7697	76	1.3499
2	0.0301	27	0.3192	52	0.7873	77	1.3703
3	0.0326	28	0.3252	53	0.8045	78	1.4023
4	0.0631	29	0.3426	54	0.8095	79	1.4090
5	0.0774	30	0.3714	55	0.8351	80	1.4172
6	0.0825	31	0.4336	56	0.8622	81	1.4193
7	0.0859	32	0.4390	57	0.8637	82	1.4224
8	0.1022	33	0.4882	58	0.8649	83	1.4384
9	0.1049	34	0.4889	59	0.8884	84	1.4897
10	0.1241	35	0.5377	60	0.8886	85	1.4916
11	0.1649	36	0.5525	61	1.0347	86	1.5326
12	0.1774	37	0.6156	62	1.0616	87	1.5442
13	0.1873	38	0.6277	63	1.0689	88	1.5877
14	0.1924	39	0.6669	64	1.0891	89	1.6302
15	0.1961	40	0.6715	65	1.0933	90	1.7115
16	0.1978	41	0.6966	66	1.1006	91	1.7947

17	0.2050	42	0.7147	67	1.1093	92	1.8339
18	0.2157	43	0.7172	68	1.1135	93	1.9330
19	0.2256	44	0.7223	69	1.1174	94	2.2588
20	0.2414	45	0.7254	70	1.1471	95	2.3505
21	0.2437	46	0.7269	71	1.1480	96	2.5855
22	0.2916	47	0.7423	72	1.1658	97	2.7694
23	0.2939	48	0.7481	73	1.2075	98	2.9443
24	0.3034	49	0.7549	74	1.2141	99	3.0349
25	0.3129	50	0.7648	75	1.3077	100	3.5784

NOTE: This concludes the results from the Ordered Statistics function found in Appendix C of TGD 2b.

FINAL VALIDATION RESULTS: The following results are for Vertical Accuracy Validation.
No spec value was supplied for vertical validation.
The lub value occurs at ordered sample 95 of 100, corresponding to a value of 2.35
The Best Estimate of the percentile is at ordered sample 90 of 100, corresponding to a value of 1.71
No max spec value was supplied for vertical validation.



B.2.2 Example 2: Demonstration of the Maximum Number of Function Inputs to the Accuracy Validation Function

Example 2: Create a set pf 100 independent 3d (x,y,z) error sample values, each component corresponds to a mean-zero Gaussian distribution with standard deviation of 1. A given 3d sample value is used to generate corresponding vertical, horizontal, and 3d radial errors. This example will demonstrate the maximum number of function inputs to the Accuracy Validation function and the resulting outputs.

```
errVals = randn(100,3);        % creates random error values
```

Error values resulting from the above command. The values are a 100-by-1 3d "vector" (actually a 100-by-3 matrix), but displayed like this to take up less space on the page. Table is distributed over two pages.

Sample Number	Sample Value			Sample Number	Sample Value		
	X	Y	Z		X	Y	Z
1	0.183432	-0.39813	-0.81931	51	-0.02809	0.984227	-0.36527
2	-0.25973	0.256436	0.967954	52	0.140359	0.993107	-0.46341
3	-1.35479	0.220025	0.030349	53	0.237586	0.668209	0.19499
4	-1.15744	-1.71144	1.351097	54	-0.67155	-0.065	-0.06695
5	-0.76199	-1.20576	-1.08637	55	-1.045	-0.6201	-0.52153
6	-0.2503	-1.77291	0.240265	56	0.965767	0.224104	0.1271
7	-1.64876	-0.07249	-1.04618	57	-0.21976	-0.46614	2.113018
8	-0.18199	-1.72111	0.618718	58	1.414536	-0.33209	0.541281
9	0.615687	0.701881	1.305034	59	-0.92419	0.924815	-0.65509
10	-0.37667	-1.00547	1.023491	60	-0.59414	1.447026	-0.86953
11	-0.69804	-1.10643	-2.11538	61	1.421275	0.595832	-0.2389
12	0.304122	1.79056	0.681729	62	1.407608	2.053271	-1.57599
13	1.10108	2.162384	0.008709	63	-1.029	-1.52925	1.825751
14	-0.44661	-0.81928	0.334273	64	0.206512	0.022688	0.204267
15	-0.46671	-0.03705	-0.54741	65	1.341111	0.095285	-1.1634
16	-1.43582	1.963004	-1.65098	66	1.332717	1.614526	0.12061
17	-0.97772	-0.54027	0.892793	67	-1.28489	0.501273	-1.44352
18	0.605895	1.717504	-0.50706	68	1.618376	-0.32384	0.582346
19	-0.11365	0.819808	-0.32979	69	0.661644	0.554217	-0.18147
20	0.764562	0.055521	0.168354	70	0.227346	0.642443	0.04201
21	0.669897	-0.35313	2.53674	71	-0.22558	0.182625	0.361842
22	0.375784	1.693149	2.293627	72	-0.96599	-2.02748	0.853715
23	0.766653	0.71105	-1.31866	73	0.095018	1.023081	-0.80977
24	1.523472	-0.63276	-0.47413	74	-0.25673	-3.49154	-1.46705
25	0.576824	0.392726	0.987382	75	2.310131	0.10949	-0.1039
26	0.425973	-0.87796	-2.07303	76	0.19006	-1.55133	-0.82291

27	-0.04293	0.148945	1.244026	77	-0.1734	-0.35511	0.424981
28	-1.24226	1.531878	-0.31524	78	-0.01403	1.398557	-1.85468
29	-0.50495	0.531848	1.399294	79	-0.61274	-0.51347	0.858575
30	0.325851	-0.75967	-0.87988	80	2.0718	1.91718	0.306394
31	-1.11142	0.347984	-0.92247	81	0.637024	0.778411	-1.11156
32	0.468184	-0.6978	2.314403	82	0.074894	-0.24646	-0.31724
33	0.322613	2.019315	0.588013	83	1.123267	-0.90353	0.837863
34	0.100021	-1.7937	-0.30977	84	-0.03313	-0.4959	2.426465
35	0.301435	-0.65951	-1.03296	85	-0.09775	0.374495	-0.35109
36	0.023824	0.771453	1.103351	86	-0.55662	-2.37053	1.381962
37	-0.02205	-0.82034	0.342109	87	-0.6155	0.615062	-0.86066
38	-0.00876	0.017888	-2.26399	88	1.604553	0.279465	0.20689
39	0.929455	0.654535	-0.90329	89	0.768495	0.818865	-0.2075
40	-0.09043	1.257699	2.970029	90	0.086933	2.047507	0.340257
41	-2.64115	-0.92708	-0.01177	91	1.704359	-0.32374	-0.49811
42	-0.48615	-0.16988	-1.99954	92	0.023634	-0.98054	-1.42139
43	0.19597	-0.47167	-0.88802	93	0.290043	1.179634	-0.27077
44	0.959722	-0.11423	-0.4199	94	-1.4199	0.894583	0.439663
45	1.380324	0.369487	0.694578	95	0.475274	-0.14293	-0.50614
46	-0.94146	-0.61862	-0.16275	96	-1.44732	-0.59347	-0.18435
47	0.760868	0.800115	0.101577	97	-0.98828	0.248917	0.401999
48	0.23792	0.426696	-0.01786	98	0.94938	-1.12978	0.539228
49	-0.20837	-0.47233	-0.96033	99	0.351224	0.066095	-0.73359
50	0.024739	0.275952	1.172806	100	-0.87228	-0.63792	-0.26837

```
% calls accuracy validation function
AccuracyValidation(errVals,'Percentile',90,'Confidence',90,'lub',1,...
                   'LEspec',3,'CEspec',4,'SEspec',5,...
                   'LEmax',12,'CEmax',16,'SEmax',20,'MinSamples',100);
```

The following is the results from the above call of the Accuracy Validation function.

The current Accuracy Validation activity is validating 100 samples.
The Percentile (Probability) Level being used is 90% (default).
The Confidence Level being used is 90% (default).
The lub (least-upper-bound) type being used is 1-sided (default).

The vertical, horizontal (radial), and 3d radial errors will be analyzed (validated).

NOTE: The following results for the vertical error samples are from the Ordered Statistics function found in Appendix C of TGD 2b.

These results contain detailed information and the user can go directly to the final validation results below if so desired.

The number of samples is 100
The k* location value is 90 Sample Value corresponding to the k* location is 1.85
The k** location value is 90.0 Sample Value corresponding to the k** location is 1.85

The k^ location value is 95 Sample Value corresponding to the k^ location is 2.26
The k^ confidence value is 93.32%
The one-sided confidence interval color code is GREEN

The k location value is 85 Sample Value corresponding to the k location is 1.44
The k+r location value is 95 Sample Value corresponding to the k+r location is 2.26
The confidence interval range is 10.10%
Determining desired confidence interval was successful.
The actual confidence percent contained within the computed confidence interval is 90.00%
The two-sided confidence interval color code is GREEN

The ordered sample values are: *Again shown this way to conserve space. Table is distributed over two pages.*

Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value
1	0.008709	26	0.317241	51	0.681729	76	1.163396
2	0.011766	27	0.329791	52	0.694578	77	1.172806
3	0.017858	28	0.334273	53	0.733593	78	1.244026
4	0.030349	29	0.340257	54	0.809772	79	1.305034
5	0.04201	30	0.342109	55	0.819308	80	1.318661
6	0.066947	31	0.351089	56	0.822913	81	1.351097
7	0.101577	32	0.361842	57	0.837863	82	1.381962
8	0.103896	33	0.365273	58	0.853715	83	1.399294
9	0.12061	34	0.401999	59	0.858575	84	1.421395
10	0.1271	35	0.419902	60	0.860655	85	1.443518
11	0.162753	36	0.424981	61	0.869529	86	1.467051
12	0.168354	37	0.439663	62	0.879881	87	1.575987
13	0.181465	38	0.463411	63	0.888015	88	1.650984
14	0.184348	39	0.474135	64	0.892793	89	1.825751
15	0.19499	40	0.498112	65	0.903286	90	1.854684
16	0.204267	41	0.506144	66	0.922472	91	1.999539
17	0.20689	42	0.507056	67	0.960327	92	2.073026
18	0.207497	43	0.52153	68	0.967954	93	2.113018
19	0.238898	44	0.539228	69	0.987382	94	2.115382

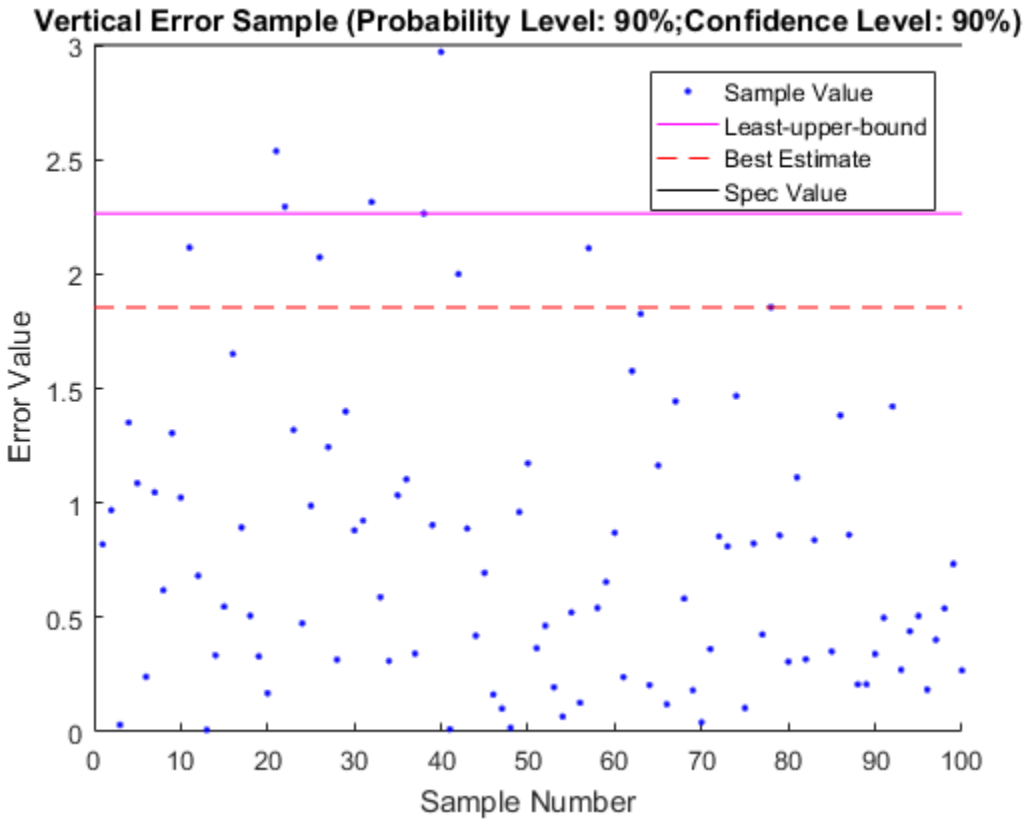
20	0.240265	45	0.541281	70	1.023491	95	2.263993
21	0.268372	46	0.547413	71	1.032964	96	2.293627
22	0.270774	47	0.582346	72	1.046181	97	2.314403
23	0.306394	48	0.588013	73	1.086374	98	2.426465
24	0.309768	49	0.618718	74	1.103351	99	2.53674
25	0.31524	50	0.655085	75	1.111562	100	2.970029

NOTE: This concludes the results from the Ordered Statistics function found in Appendix C of TGD 2b.

FINAL VALIDATION RESULTS: The following results are for Vertical Accuracy Validation.
The vertical error samples PASS validation for the spec value test and the max spec value test.

Least-Upper-Bound Value: 2.26 Spec Value: 3.00
Max Error Sample: 2.97 Max Spec Value: 12.00

The lub value occurs at ordered sample 95 of 100, corresponding to a value of 2.26
The Best Estimate of the percentile is at ordered sample 90 of 100, corresponding to a value of 1.85



NOTE: The following results for the horizontal (radial) error samples are from the Ordered Statistics function found in Appendix C of TGD 2b.

These results contain detailed information and the user can go directly to the final validation results below if so desired.

The number of samples is 100

The k* location value is 90 Sample Value corresponding to the k* location is 2.07

The k** location value is 90.0 Sample Value corresponding to the k** location is 2.07

The k^ location value is 95 Sample Value corresponding to the k^ location is 2.43

The k^ confidence value is 93.32%

The one-sided confidence interval color code is GREEN

The k location value is 85 Sample Value corresponding to the k location is 1.82

The k+r location value is 95 Sample Value corresponding to the k+r location is 2.43

The confidence interval range is 10.10%

Determining desired confidence interval was successful.

The actual confidence percent contained within the computed confidence interval is 90.00%

The two-sided confidence interval color code is GREEN

The ordered sample values are: *Again shown this way to conserve space. Table is distributed over two pages.*

Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value
1	0.019917	26	0.757271	51	1.117057	76	1.650356
2	0.155009	27	0.766575	52	1.122998	77	1.650458
3	0.207755	28	0.771821	53	1.126516	78	1.678211
4	0.257583	29	0.79944	54	1.136795	79	1.73071
5	0.277059	30	0.820638	55	1.164626	80	1.734349
6	0.290235	31	0.826603	56	1.214768	81	1.734834
7	0.357389	32	0.827649	57	1.215133	82	1.790495
8	0.364993	33	0.840313	58	1.260946	83	1.796486
9	0.387041	34	0.863092	59	1.307442	84	1.816204
10	0.395187	35	0.870138	60	1.308218	85	1.821243
11	0.438352	36	0.933099	61	1.344492	86	1.843217
12	0.468181	37	0.933653	62	1.372539	87	1.972275
13	0.488544	38	0.966497	63	1.379213	88	2.044923
14	0.4963	39	0.975843	64	1.398628	89	2.049351
15	0.497008	40	0.980827	65	1.426348	90	2.066083
16	0.510763	41	0.984628	66	1.428921	91	2.093521
17	0.514976	42	0.991427	67	1.441561	92	2.245841
18	0.515341	43	1.002977	68	1.452995	93	2.312725
19	0.516252	44	1.005844	69	1.475713	94	2.426578
20	0.674687	45	1.019145	70	1.541116	95	2.43207

21	0.681484	46	1.027484	71	1.562931	96	2.435005
22	0.697825	47	1.045633	72	1.564253	97	2.489434
23	0.70919	48	1.073708	73	1.56427	98	2.79913
24	0.72513	49	1.080659	74	1.628709	99	2.822753
25	0.733375	50	1.104131	75	1.649653	100	3.500968

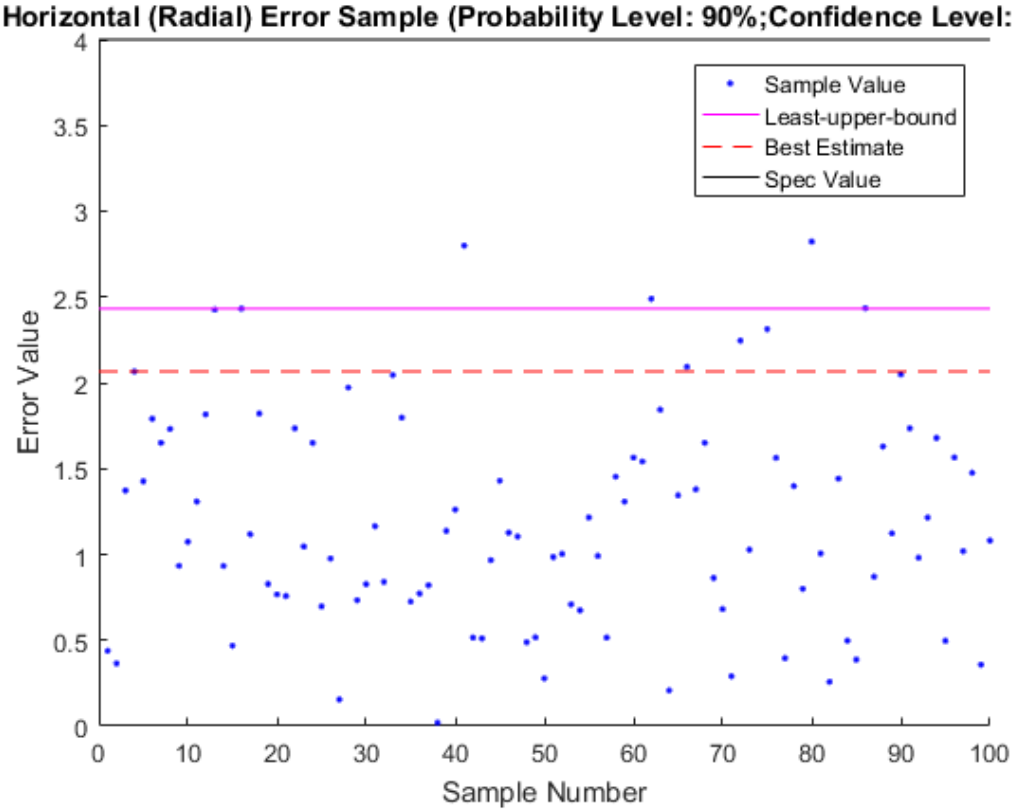
NOTE: This concludes the results from the Ordered Statistics function found in Appendix C of TGD 2b.

FINAL VALIDATION RESULTS: The following results are for Horizontal (radial) Accuracy Validation.
The horizontal (radial) error samples PASS validation for the spec value test and the max spec value test.

Least-Upper-Bound Value: 2.43 Spec Value: 4.00

Max Error Sample: 3.50 Max Spec Value: 16.00

The lub value occurs at ordered sample 95 of 100, corresponding to a value of 2.43
The Best Estimate of the percentile is at ordered sample 90 of 100, corresponding to a value of 2.07



NOTE: The following results for the 3d radial error samples are from the Ordered Statistics function found in Appendix C of TGD 2b.

These results contain detailed information and the user can go directly to the final validation results below if so desired.

The number of samples is 100

The k* location value is 90 Sample Value corresponding to the k* location is 2.49

The k** location value is 90.0 Sample Value corresponding to the k** location is 2.49

The k^ location value is 95 Sample Value corresponding to the k^ location is 2.84

The k^ confidence value is 93.32%

The one-sided confidence interval color code is GREEN

The k location value is 85 Sample Value corresponding to the k location is 2.40

The k+r location value is 95 Sample Value corresponding to the k+r location is 2.84

The confidence interval range is 10.10%

Determining desired confidence interval was successful.

The actual confidence percent contained within the computed confidence interval is 90.00%

The two-sided confidence interval color code is GREEN

The ordered sample values are: *Again shown this way to conserve space. Table is distributed over two pages.*

Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value	Sample Number	Sample Value
1	0.291354	26	1.104859	51	1.571145	76	2.06479
2	0.408645	27	1.108793	52	1.575095	77	2.077406
3	0.46386	28	1.113484	53	1.57983	78	2.096992
4	0.488871	29	1.138212	54	1.58879	79	2.127785
5	0.522556	30	1.142007	55	1.604625	80	2.174954
6	0.580329	31	1.173139	56	1.641796	81	2.264081
7	0.678	32	1.205087	57	1.667367	82	2.291224
8	0.682777	33	1.207255	58	1.682919	83	2.315057
9	0.708869	34	1.209084	59	1.716438	84	2.322932
10	0.720315	35	1.223874	60	1.726958	85	2.40263
11	0.735508	36	1.24458	61	1.734847	86	2.426594
12	0.784844	37	1.253646	62	1.750183	87	2.462232
13	0.816018	38	1.262073	63	1.766335	88	2.468636
14	0.881963	39	1.308225	64	1.777962	89	2.476843
15	0.889092	40	1.322324	65	1.789683	90	2.487223
16	0.890935	41	1.346511	66	1.792952	91	2.594382
17	0.929203	42	1.372875	67	1.804928	92	2.647359

18	0.991167	43	1.429999	68	1.806544	93	2.799155
19	0.999541	44	1.451974	69	1.822997	94	2.799833
20	1.024427	45	1.462375	70	1.837979	95	2.839333
21	1.034483	46	1.483369	71	1.890511	96	2.875533
22	1.050198	47	1.485702	72	1.939935	97	2.939509
23	1.053771	48	1.499097	73	1.954014	98	2.946356
24	1.090295	49	1.550541	74	1.996491	99	3.226617
25	1.095564	50	1.559523	75	1.997309	100	3.795921

NOTE: This concludes the results from the Ordered Statistics function found in Appendix C of TGD 2b.

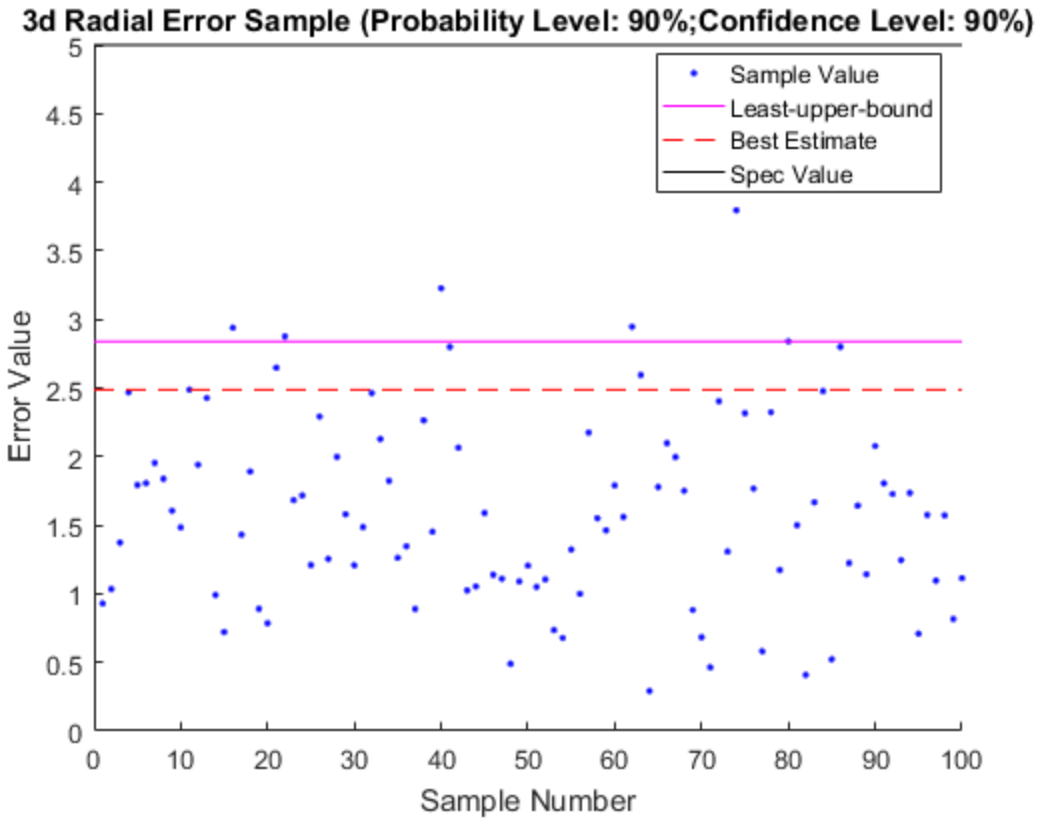
FINAL VALIDATION RESULTS: The following results are for 3d Radial Accuracy Validation.
The 3d radial error samples PASS validation for the spec value test and the max spec value test.

Least-Upper-Bound Value: 2.84 Spec Value: 5.00

Max Error Sample: 3.80 Max Spec Value: 20.00

The lub value occurs at ordered sample 95 of 100, corresponding to a value of 2.84

The Best Estimate of the percentile is at ordered sample 90 of 100, corresponding to a value of 2.49



Appendix C: Predicted Accuracy Validation Pseudo-code

This Appendix contains pseudo-code to perform Validation of Predicted Accuracy Requirements. It performs the entire validation process per Sections 4.2/5.2, given the appropriate inputs as documented below.

Note that although the code may appear somewhat complicated/detailed, the underlying algorithms is not. The majority of the code involves accommodating optional inputs, error checking and warnings associated with the inputs, and formatting the output which includes plots. The code is literally a "drop in" to a MATLAB capability.

In addition, following the pseudo-code are examples of function calls with corresponding inputs and computed outputs.

C.1 Pseudo-code

```
function PredictedAccuracyValidation(errVal,varargin)

%%%%% The "PredictedAccuracyValidation" function validates predicted accuracy
%%%%% using a set of error samples or values (errVal) and corresponding
%%%%% predicted error covariance matrices (or their approximate equivalent)
%%%%% supplied by the user.
%%%%%
%%%%% The algorithm/equations corresponding to this function described in
%%%%% the document "Accuracy and Predicted Accuracy in the NSG:
%%%%% Specification and Validation; Technical Guidance Document (TGD) 2c".
%%%%%
%%%%% Inputs: errVal = the error samples or values supplied by the user.
%%%%%                  This variable is a matrix consisting of n (number of
%%%%%                  samples) rows and 1, 2, or 3 columns resulting in
%%%%%                  the validation of the vertical, horizontal (radial),
%%%%%                  or 3d radial error, respectively.  This input is
%%%%%                  required and is always the first input into this
%%%%%                  function.  If a three column matrix is supplied all
%%%%%                  three (vertical, horizontal (radial), and 3d radial)
%%%%%                  accuracy validation calculations will be performed.
%%%%%                  The minimum number of samples should be 100.
%%%%%
%%%%%         varargin = this a variable number of inputs that could be
%%%%%                    entered into this function.  The list of possible
%%%%%                    inputs follows.  One of the predicted accuracy
%%%%%                    options is required as an input.  None of the
%%%%%                    other variables are required as inputs because the
%%%%%                    function has default values set.  After each
%%%%%                    possible input definition below will be how the
%%%%%                    input should be called for use.
%%%%%
```

```
%%%%%                   predicted accuracy covariance (predCov) = the
%%%%%                      predicted error covariances supplied by the
%%%%%                      user.  This variable is a matrix consisting of
%%%%%                      n (number of samples) covariances.  When only
%%%%%                      the vertical error is being validated this
%%%%%                      matrix will be a n-by-1 vector of the variance
%%%%%                      corresponding to the vertical component of the
%%%%%                      entire covariance matrix.  When the horizontal
%%%%%                      (radial) error is being validated this matrix
%%%%%                      will be a 2-by-2-by-n matrix corresponding to
%%%%%                      the horizontal components of the entire
%%%%%                      covariance matrix.  Finally, when the 3d radial
%%%%%                      error is being validated this matrix will be a
%%%%%                      3-by-3-by-n matrix.  This input is required and
%%%%%                      is always the second input into this function.
%%%%%                      If a three column matrix is supplied in
%%%%%                      variable "errVal" and a 3-by-3-by-n matrix is
%%%%%                      supplied in this matrix all three (vertical,
%%%%%                      horizontal (radial), and 3d radial) predicted
%%%%%                      accuracy validation calculations will be
%%%%%                      performed.  This is the prefered method for
%%%%%                      predicted accuracy validation.  To set the
%%%%%                      predicted accuracy covariance the user would
%%%%%                      enter:
%%%%%                      "'PredictedCovariance',[covariance matrix]".
%%%%%
%%%%%                   computed predicted scalar accuracy (scalarCov) =
%%%%%                      the predicted error covariances supplied by the
%%%%%                      user will be used to calculate the
%%%%%                      corresponding LE, CE, and SE values.  The error
%%%%%                      covariance should be of the same format as
%%%%%                      described above under "predicted accuracy
%%%%%                      covariance".  To set the computed predicted
%%%%%                      scalar accuracy the user would enter:
%%%%%                      "'ComputedScalar',[covariance matrix]".
%%%%%
%%%%%                   entered predicted scalar accuracy (scalarVal) =
%%%%%                      the predicted error values supplied by the user
%%%%%                      will be a n-by-1 vector of CE90 values.  This
%%%%%                      option will only be for horizontal (radial)
%%%%%                      errors at a probability level of 90%.  To set
%%%%%                      the entered predicted scalar accuracy the user
%%%%%                      would enter: "'EnteredScalar',[CE90 vector]".
%%%%%
%%%%%                   fidelity level (fidLev) = the desired fidelity
%%%%%                      level for the validation.  The three possible
%%%%%                      levels are: high, medium, and low.  These
%%%%%                      levels correspond to the normalized error test
%%%%%                      tolerance table.  The default value for the
%%%%%                      fidelity level is high.  If the user does not
%%%%%                      input a value the default value will be used.
%%%%%                      To set the fidelity level the user would enter:
%%%%%                      "'FidelityLevel','medium'".
%%%%%
```

```
%%%%%                   minimum number of i.i.d. error samples (minSamp) =
%%%%%                      minimum number of independent sample values for
%%%%%                      testing validation as specified for the current
%%%%%                      system/process being validated.  To set the
%%%%%                      minimum number of i.i.d. error samples value
%%%%%                      the user would enter "'MinSamples',50".
%%%%%
%%%%%                   normalized error test at 0.999999 probability
%%%%%                      level (test999999) = optional test at the
%%%%%                      probability level 0.999999.  To set this option
%%%%%                      the user would enter "'Test0.999999','on'".
%%%%%
%%%%%                   vertical normalized error tolerance (VaccFid) =
%%%%%                      table of three probability levels (99, 90, and
%%%%%                      50%) as a function of both the number of
%%%%%                      samples for the tests (statistical
%%%%%                      significance), and the applicable level of
%%%%%                      predicted accuracy fidelity. Table 5.4.2.3 in
%%%%%                      Section 5.4 of TGD 2c is the default values
%%%%%                      used unless this optional input is supplied.
%%%%%                      This input should take the same format as Table
%%%%%                      5.4.2.3, being a 9-by-4 matrix with columns
%%%%%                      corresponding to number of samples 400, 100,
%%%%%                      50, and 25.  The rows correspond to 99, 90, and
%%%%%                      50% probability levels for the high, medium,
%%%%%                      and low fidelity levels.  To set this option
%%%%%                      the user would enter "'VerticalFidelity',
%%%%%                      [vertical fidelity matrix]".
%%%%%
%%%%%                   horizontal normalized error tolerance (HaccFid) =
%%%%%                      table of three probability levels (99, 90, and
%%%%%                      50%) as a function of both the number of
%%%%%                      samples for the tests (statistical
%%%%%                      significance), and the applicable level of
%%%%%                      predicted accuracy fidelity. Table 5.4.2.2 in
%%%%%                      Section 5.4 of TGD 2c is the default values
%%%%%                      used unless this optional input is supplied.
%%%%%                      This input should take the same format as Table
%%%%%                      5.4.2.2, being a 9-by-4 matrix with columns
%%%%%                      corresponding to number of samples 400, 100,
%%%%%                      50, and 25.  The rows correspond to 99, 90, and
%%%%%                      50% probability levels for the high, medium,
%%%%%                      and low fidelity levels.  To set this option
%%%%%                      the user would enter "'HorizontalFidelity',
%%%%%                      [horizontal fidelity matrix]".
%%%%%
```

```
%%%%%                     3d radial normalized error tolerance (RaccFid) =
%%%%%                        table of three probability levels (99, 90, and
%%%%%                        50%) as a function of both the number of
%%%%%                        samples for the tests (statistical
%%%%%                        significance), and the applicable level of
%%%%%                        predicted accuracy fidelity. Table 5.4.2.4 in
%%%%%                        Section 5.4 of TGD 2c is the default values
%%%%%                        used unless this optional input is supplied.
%%%%%                        This input should take the same format as Table
%%%%%                        5.4.2.4, being a 9-by-4 matrix with columns
%%%%%                        corresponding to number of samples 400, 100,
%%%%%                        50, and 25.  The rows correspond to 99, 90, and
%%%%%                        50% probability levels for the high, medium,
%%%%%                        and low fidelity levels.  To set this option
%%%%%                        the user would enter "'RadialFidelity',
%%%%%                        [3d radial fidelity matrix]".
%%%%%
%%%%% Outputs: Statements of the results are printed to the Command Window.
%%%%%
%%%%%          Figures of sample error values vs. predicted percentile
%%%%%          ellipsoid radial distance are plotted.

%%%%% CHECKING INPUTS
%%% Checks that the error sample value matrix is oriented correctly.
errSize = size(errVal);              % checks size of error value matrix
if errSize(1) < errSize(2)           % compares number of rows to columns
    errVal  = transpose(errVal);     % transposes error value matrix
end                                  % ends loop comparing rows and columns
if size(errVal,2) > 3                % checks number of error components
    fprintf('\n\nThis Accuracy Validation function does not handle data ')
    fprintf('sets larger than 3d.\n\n') % prints statement to command window
    return                           % exits function
end                                  % ends loop checking error components
if size(errVal,1) < 25               % checks number of error sample values
    fprintf('\n\nThe number of Error Samples entered (%i ',size(errVal,1))
    fprintf('< 25) is not large enough for calculations to be ')
    fprintf('performed.\n')          % prints statement to command window
    return                           % exits function
elseif size(errVal,1) < 50           % checks number of error sample values
    %%% Prints warning to command window
    fprintf('\n\nWARNING: The number of Error Samples entered ')
    fprintf('(%i < 50) is less than the recommended minimum ',size(errVal,1))
    fprintf('number of samples.  Processing will continue, but results ')
    fprintf('should be taken under careful consideration.\n')
end                                  % ends loop checking number of values

predCov     = [];                    % creates accuracy covariance matrix
scalarCov   = [];                    % creates calced scalar accuracy variable
scalarVal   = [];                    % creates input scalar accuracy variable
fidLev      = 'high';                % sets default fidelity level value
minSamp     = [];                    % creates minimum sample variable
test999999  = 'off';                 % creates 99.9999% test flag variable
```

```
VaccFid = [97 95 94 90;86 83 80 76;44 40 38 34
           96 91 90 86;80 78 74 70;70 34 32 30
           88 86 84 80;70 66 64 62;30 28 26 22]; % vertical fidelity
HaccFid = [97 95 93 90;85 83 78 76;44 39 38 34
           95 90 88 84;78 76 72 68;36 30 30 24
           85 84 82 76;65 64 60 54;25 24 18 14]; % horizontal fidelity
RaccFid = [96 94 93 87;84 82 78 73;42 37 37 31
           89 85 84 82;72 71 69 66;32 27 27 22
           80 79 79 72;59 57 57 50;19 15 15 10]; % 3d radial fidelity
if nargin < 3                          % checks number of function inputs
    fprintf('\n\nThe number of inputs into the function are not enough for ')
    fprintf('predicted accuracy validation.\nPlease enter the predicted ')
    fprintf('covariance, computed predicted scalar accuracy covariance, or ')
    fprintf('the predicted scalar accuracy CE values.\n\n')
    return                             % exits function
else                                   % checks number of function inputs
    for n = 1:2:nargin-2               % starts loop to cycle through inputs
        if strcmp(varargin{n},'PredictedCovariance') || ...
           strcmp(varargin{n},'ComputedScalar') % checks input type
            if isnumeric(varargin{n+1})  % checks entered value is a number
                predCov = varargin{n+1}; % defines predicted covariance
            else                       % checks entered value is a number
                fprintf('\n\nThe predicted accuracy function was expecting ')
                fprintf('a numeric matrix.\nPlease check covariance ')
                fprintf('input.\n\n') % prints statement to command window
                return                 % exits function
            end                        % ends loop checking entered value

            if size(errVal,2) == 1  % checks if validating vertical
                if size(predCov,1) == 1 % checks size of predicted covariance
                    predCov  = transpose(predCov); % changes direction
                end                    % ends loop checking size of covariance
                if size(predCov,1) ~= size(errVal,1) % variances to errors
                    %%% Prints statement to command window
                    fprintf('\n\nWARNING: The number of Predicted ')
                    fprintf('Variances entered (%i) does ',size(predCov,1))
                    fprintf('not equal the number of Error Samples entered ')
                    fprintf('(%i).  Please enter the ',size(errVal,1))
                    fprintf('same number of error values and variances.\n')
                    return             % exits function
                end                    % ends comparison of variances to errors
            elseif size(errVal,2) == 2  % checks if validating horizontal
                if size(predCov,1) == 1 % checks predicted covariance matrix
                    %%% Prints statement to command window
                    fprintf('\n\nTo analyze the horizontal (radial) ')
                    fprintf('component the entered Predicted Covariance ')
                    fprintf('matrix should be 2-by-2-by-%i\n',size(errVal,1))
                    return             % exits function
```

```
elseif size(predCov,1) == 3 % checks size of covariance
    if size(predCov,2) == 3 % checks size of covariance
        if size(predCov,3) ~= size(errVal,1) % checks covar
            %%% Prints statement to command window
            fprintf('\n\nWARNING: The number of Predicted ')
            fprintf('Covariances entered ')
            fprintf('(%i) does not equal ',size(predCov,3))
            fprintf('the number of Error Samples entered ')
            fprintf('(%i).  Please enter ',size(errVal,1))
            fprintf('the same number of error values and ')
            fprintf('covariances.\n')
            return  % exits function
        else        % checks size of predicted covariance
            fprintf('The upper left 2-by-2 protion of the ')
            fprintf('Predicted Covariance will be used.\n')
            predCov = predCov(1:2,1:2,:); % reduces size
        end         % ends loop checking size
    else            % checks size of predicted covariance
        fprintf('\n\nThe entered Predicted Covariance ')
        fprintf('matrix should be 2-by-2-by-')
        fprintf('%i\n',size(errVal,1))  % prints statement
        return      % exits function
    end             % ends loop checking size of covariance
elseif size(predCov,1) == 2 % checks size of covariance
    if size(predCov,2) == 2 % checks size of covariance
        if size(predCov,3) ~= size(errVal,1) % checks covar
            %%% Prints statement to command window
            fprintf('\n\nWARNING: The number of Predicted ')
            fprintf('Covariances entered')
            fprintf('(%i) does not equal ',size(predCov,3))
            fprintf('the number of Error Samples entered ')
            fprintf('(%i). Please enter ',size(errVal,1))
            fprintf('the same number of error values and ')
            fprintf('covariances.\n')
            return  % exits function
        end         % ends loop checking size of covariance
    else            % checks size of predicted covariance
        fprintf('\n\nThe entered Predicted Covariance ')
        fprintf('matrix should be 2-by-2-by-')
        fprintf('%i\n',size(errVal,1))  % prints statement
        return      % exits function
    end             % ends loop checking size of covariance
else                % checks size of predicted covariance
    fprintf('\n\nThe entered Predicted Covariance matrix ')
    fprintf('should be 2-by-2-by-%i\n',size(errVal,1))
    return          % exits function
end                 % ends loop checking size of covariance
```

```
    elseif size(errVal,2) == 3  % checks if validating 3d radials
        if size(predCov,1) == 3 % checks size of predicted covariance
            if size(predCov,2) == 3 % checks size of covariance
                if size(predCov,3) ~= size(errVal,1) % checks covar
                    fprintf('\n\nWARNING: The number of Predicted ')
                    fprintf('Covariances entered ')
                    fprintf('(%i) does not equal ',size(predCov,3))
                    fprintf('the number of Error Samples entered ')
                    fprintf('(%i). Please enter ',size(errVal,1))
                    fprintf('the same number of error values and ')
                    fprintf('covariances.\n')
                    return  % exits function
                end         % ends loop checking size of covariance
            else            % checks size of predicted covariance
                fprintf('\n\nThe entered Predicted Covariance ')
                fprintf('matrix should be 3-by-3-by-')
                fprintf('%i\n',size(errVal,1)) % prints statement
                return      % exits function
            end             % ends loop checking size of covariance
        else                % checks size of predicted covariance
            fprintf('\n\nThe entered Predicted Covariance matrix ')
            fprintf('should be 3-by-3-by-%i\n',size(errVal,1))
            return          % exits function
        end                 % ends loop checking size of covariance
    end                     % ends loop checking type of validation
    if strcmp(varargin{n},'ComputedScalar') % checks input type
        scalarCov   = predCov;  % computed predicted scalar accuracy
        predCov     = [];       % resets predicted covariance
    end                     % ends loop checking input type
elseif strcmp(varargin{n},'EnteredScalar') % checks input type
    if isnumeric(varargin{n+1})  % checks entered value is a number
        scalarVal = varargin{n+1}; % defines predicted covariance
        if size(errVal,2) == 3  % checks components of error values
            fprintf('\n\nOnly the horizontal portion of the ')
            fprintf('entered error values will be used.\n')
            errVal  = errVal(:,1:2); % reduces error value matrix
        end                 % ends loop checking error values
    else                    % checks entered value is a number
        fprintf('\n\nThe predicted accuracy function was expecting ')
        fprintf('a numeric vector.\nPlease check input predicted ')
        fprintf('scalar accuracy input.\n\n') % prints statement
        return              % exits function
    end                     % ends loop checking entered value

    if size(scalarVal,1) == 1 % checking orientation of vector
        scalarVal   = transpose(scalarVal); % reversing orientation
    end
```

```
            if size(scalarVal,1) ~= size(errVal,1) % compares CE90s to errors
                %%% Prints statement to command window
                fprintf('\n\nWARNING: The number of Predicted Scalar ')
                fprintf('Accuracy values entered (%i) ',size(scalarVal,1))
                fprintf('does not equal the number of Error Samples ')
                fprintf('entered (%i).  Please enter the ',size(errVal,1))
                fprintf('same number of error values and CE90 values.\n')
                return              % exits function
            end                     % ends comparison of CE90s to errors
        elseif strcmp(varargin{n},'FidelityLevel') % checks input type
            if isnumeric(varargin{n+1}) == 0 % checks entered value is number
                fidLev  = varargin{n+1};    % fidelity level
            end                     % ends loop checking entered value
        elseif strcmp(varargin{n},'MinSamples') %checks min number of samples
            if isnumeric(varargin{n+1})     % checks entered value is number
                minSamp = varargin{n+1};    % min number of error samples
            end                     % ends loop checking entered value
        elseif strcmp(varargin{n},'Test0.999999') % checks for 99.9999% test
            if isnumeric(varargin{n+1}) == 0 % checks entered value is number
                test999999  = varargin{n+1}; % updates 99.9999% test value
            end                     % ends loop checking entered value
        elseif strcmp(varargin{n},'VerticalFidelity') % checks vert fidelity
            if isnumeric(varargin{n+1}) % checks entered value is number
                if size(varargin{n+1}) == [9 4] % checks size of input
                    VaccFid = varargin{n+1}; % updates vertical fidelity
                else                % checks size of input
                    fprintf('\n\nInput vertical fidelity matrix is the ')
                    fprintf('incorrect size.  Default matrix is being ')
                    fprintf('used.\n') % prints statement to command window
                end                 % ends loop checking input size
            end                     % ends loop checking entered value
        elseif strcmp(varargin{n},'HorizontalFidelity') % checks hor fidelity
            if isnumeric(varargin{n+1}) % checks entered value is number
                if size(varargin{n+1}) == [9 4] % checks size of input
                    HaccFid = varargin{n+1}; % updates vertical fidelity
                else                % checks size of input
                    fprintf('\n\nInput horizontal fidelity matrix is the ')
                    fprintf('incorrect size.  Default matrix is being ')
                    fprintf('used.\n') % prints statement to command window
                end                 % ends loop checking input size
            end                     % ends loop checking entered value
        elseif strcmp(varargin{n},'RadialFidelity') % checks radial fidelity
            if isnumeric(varargin{n+1}) % checks entered value is number
                if size(varargin{n+1}) == [9 4] % checks size of input
                    RaccFid = varargin{n+1}; % updates vertical fidelity
                else                % checks size of input
                    fprintf('\n\nInput 3d radial fidelity matrix is the ')
                    fprintf('incorrect size.  Default matrix is being ')
                    fprintf('used.\n') % prints statement to command window
                end                 % ends loop checking input size
            end                     % ends loop checking entered value
        end                         % ends loop checking input type
    end                             % ends loop cycling through inputs
end                                 % ends loop checking number of inputes

numSamp = size(errVal,1);           % number of error samples being validated
```

```
fprintf('\n\nThe current Predicted Accuracy Validation activity is ')
fprintf('validating %i samples.\n',numSamp) % prints statement
fprintf('The validation will be performed using the ') % prints statement
if isempty(predCov) == 0            % checks type of validation
    fprintf('predicted radials.\n') % prints statement to command window
elseif isempty(scalarCov) == 0      % checks type of validation
    fprintf('predicted scalar accuracy metrics computed from error ')
    fprintf('covariance matrices.\n') % prints statement to command window
elseif isempty(scalarVal) == 0      % checks type of validation
    fprintf('predicted scalar accuracy metrics entered by the user.\n')
end                                 % ends loop checking type of validation
fprintf('The Fidelity Level being used is %s',fidLev) % prints statement
if strcmp(fidLev,'high')            % checks fidelity level being used
    fprintf(' (default).\n')        % prints statement to command window
else                                % checks fidelity level being used
    fprintf('.\n')                  % prints statement to command window
end                                 % ends loop checking probability level
if isempty(minSamp) == 0            % checks min number of error samples
    if numSamp < minSamp            % checks if number of samples is met
        fprintf('WARNING: The number of error samples (%i) ',numSamp)
        fprintf('is less than the minimum number of i.i.d. error ')
        fprintf('samples (%i).\n',minSamp)  % prints statement
    else                            % checks if number of samples is met
        fprintf('The number of error samples (%i) meets the ',numSamp)
        fprintf('minimum number of i.i.d. error samples (%i).\n',minSamp)
    end                             % ends loop checking number of samples
end                                 % ends loop checking min number of i.i.d.
%%%%% CHECKING INPUTS

%%% Interpolation between number of samples for use with tables.
sampLev = [400 100 50 25];          % number of sample values
if numSamp >= 400                   % checks number of samples
    levSc   = 1;                    % sets sample level scale
elseif numSamp == 25                % checks number of samples
    levSc   = 4;                    % sets sample level scale
else                                % checks number of samples
    l1      = find(sampLev >= numSamp); % sample levels above sample number
    l2      = find(sampLev < numSamp);  % sample levels below sample number
    levSc   = l1(end)+((sampLev(l1(end))-numSamp)/(sampLev(l1(end))-...
              sampLev(l2(1))));     % sets sample level scale
end                                 % ends loop checking number of samples

predMag = [5.0 2.576 1.645 0.674
           5.3 3.035 2.146 1.177
           5.5 3.368 2.500 1.538];  % predicted magnitude values

%%% Use of the Predictive Statistics function as found in "Accuracy and
%%% Predicted Accuracy in the NSG: Predictive Statistics; Technical
%%% Guidance Document TGD 2a Section 7: Reference Appendix C.3 Pseudo-code".
%%% Example function call:
%%% LEcal = LEintegral(covar,mCoord,prob)
%%% CEcal = CEintegral(covar,mCoord,prob)
%%% SEcal = SEintegral(covar,mCoord,prob)
```

```
%%%%% Calculations being performed
if size(errVal,2) == 1 || size(errVal,2) == 3 % checks if testing vertical
    if size(errVal,2) == 1          % checks if testing 1d or 3d
        fprintf('\n\nThe vertical component of the sample errors will be ')
        fprintf('analyzed (validated).\n\n')    % prints statement
        verErr  = abs(errVal);      % vertical error values
        if isempty(predCov) == 0    % checks type of validation
            fprintf('\nThe vertical component is being analyzed ')
            fprintf('(validated) with the predicted radials.\n')
            verCov  = predCov;      % vertical error variances
        elseif isempty(scalarCov) == 0 % checks type of validation
            fprintf('\nThe vertical component is being analyzed ')
            fprintf('(validated) with the computed predicted scalar ')
            fprintf('accuracy metrics.\n') % prints statement
            verCov  = scalarCov;    % vertical error variances
        else                        % checks type of validation
            fprintf('\nAn error has occured in the determination of the ')
            fprintf('type of validation to apply to the vertical ')
            fprintf('component.\n') % prints statement to command window
            return                  % exits function
        end                         % ends loop checking type of validation
    else                            % checks if testing 1d or 3d
        fprintf('\n\nThe vertical, horizontal (radial), and 3d radial ')
        fprintf('components of the sample errors will be analyzed ')
        fprintf('(validated).\n\n') % prints statement to command window
        verErr  = abs(errVal(:,3)); % vertical error values
        if isempty(predCov) == 0    % checks type of validation
            fprintf('\nThe vertical component is being analyzed ')
            fprintf('(validated) with the predicted radials.\n')
            verCov  = reshape(predCov(3,3,:),size(verErr)); % vertical covar
        elseif isempty(scalarCov) == 0 % checks type of validaiton
            fprintf('\nThe vertical component is being analyzed ')
            fprintf('(validated) with the computed predicted scalar ')
            fprintf('accuracy metrics.\n') % prints statement
            verCov  = reshape(scalarCov(3,3,:),size(verErr)); % covariance
        else                        % checks type of validation
            fprintf('\nAn error has occured in the determination of the ')
            fprintf('type of validation to apply to the vertical ')
            fprintf('component.\n') % prints statement to command window
            return                  % exits function
        end                         % ends loop checking type of validation
    end                             % ends loop checking tests to perform

    sampErr = verErr*ones(1,4);     % sample radial error
    if isempty(predCov) == 0        % checks validation type
        normNum = sqrt(verErr.*(1./verCov).*verErr); % normalized numerator
        verNorm = (normNum*ones(1,4))./...
                  (ones(numSamp,1)*predMag(1,:)); % vert normalized errors
        predRad = verErr*predMag(1,:).*...
                  (((verErr.*(1./verCov).*verErr).^...
                  (-1/2))*ones(1,4)); % predicted sample radial error
```

```
elseif isempty(scalarCov) == 0  % checks validation type
    prob    = [.99 .90 .50];    % probability levels for tests
    predRad = zeros(numSamp,4); % creates matrix for LE values
    fprintf('\tCalculating LE values for probability ')
    for m = 1:length(prob)      % starts loop to cycle thru probabilities
        fprintf('%i%% ... ',prob(m)*100)    % prints statement
        for n = 1:numSamp       % starts loop to cycle through samples
            predRad(n,m+1) = LEintegral(verCov(n),0,prob(m)); % LE values
        end                     % ends loop cycling through samples
    end                         % ends loop cycling thru probabilities
    fprintf('done.\n\n')        % prints statement to command window

    if strcmp(test999999,'on')  % checks if performing 99.9999% test
        predRad(:,1) = predMag(1,1)/predMag(1,3)*predRad(:,3); % LE999999
    end                         % ends loop checking 99.9999% test

    verNorm = sampErr./predRad; % check normalized errors
end                             % ends loop checking validation type

req9999 = 100;                  % sets pass percent for 99.9999% level
%%% Interpolation for pass percentage from table
if floor(levSc) == ceil(levSc)  % checks interpolation position
    if strcmp(fidLev,'high')    % checks if using high fidelity
        req99   = VaccFid(1,floor(levSc)); % 99% level
        req90   = VaccFid(2,floor(levSc)); % 90% level
        req50   = VaccFid(3,floor(levSc)); % 50% level
    elseif strcmp(fidLev,'medium')  % checks if using medium fidelity
        req99   = VaccFid(4,floor(levSc)); % 99% level
        req90   = VaccFid(5,floor(levSc)); % 90% level
        req50   = VaccFid(6,floor(levSc)); % 50% level
    else                        % checks if using low fidelity
        req99   = VaccFid(7,floor(levSc)); % 99% level
        req90   = VaccFid(8,floor(levSc)); % 90% level
        req50   = VaccFid(9,floor(levSc)); % 50% level
    end                         % ends loop checking fidelity
else                            % checks interpolation position
    if strcmp(fidLev,'high')    % checks if using high fidelity
        req99   = VaccFid(1,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(1,floor(levSc))-...
                  VaccFid(1,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = VaccFid(2,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(2,floor(levSc))-...
                  VaccFid(2,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = VaccFid(3,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(3,floor(levSc))-...
                  VaccFid(3,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
```

```
    elseif strcmp(fidLev,'medium')  % checks if using medium fidelity
        req99   = VaccFid(4,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(4,floor(levSc))-...
                  VaccFid(4,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = VaccFid(5,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(5,floor(levSc))-...
                  VaccFid(5,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = VaccFid(6,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(6,floor(levSc))-...
                  VaccFid(6,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    else                          % checks if using low fidelity
        req99   = VaccFid(7,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(7,floor(levSc))-...
                  VaccFid(7,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = VaccFid(8,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(8,floor(levSc))-...
                  VaccFid(8,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = VaccFid(9,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(VaccFid(9,floor(levSc))-...
                  VaccFid(9,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    end                           % ends loop checking fidelity
end                               % ends loop checking table position

fprintf('FINAL VALIDATION RESULTS: The following results are for ')
fprintf('Vertical Accuracy Validation.\n') % prints statement

if strcmp(test999999,'on')        % checks if performing 99.9999% test
    num9999 = length(find(verNorm(:,1) <= 1)); % 99.9999% passing number
    act9999 = num9999/numSamp*100;  % percent passing 99.9999% level test
    fprintf('%i of %i samples passed the 99.9999%% ',num9999,numSamp)
    fprintf('validation level test\n') % prints statement
    fprintf('\tRequired Percentage:%6.1f\t',req9999) % prints statement
    fprintf('\tActual Percentage:%6.1f\t',act9999)   % prints statament
    fprintf('\tValidation Result: ') % prints statement to command window
    if act9999 >= req9999         % checks if test is passed
        fprintf('PASS\n')         % prints statement to command window
    else                          % checks if test is failed
        fprintf('FAIL\n')         % prints statement to command window
    end                           % ends loop checking test status
end                               % ends loop checking for 99.9999% test

num99   = length(find(verNorm(:,2) <= 1)); % 99% passing number
act99   = num99/numSamp*100;      % percent passing 99% level test
fprintf('%i of %i samples passed the 99%% ',num99,numSamp)
fprintf('validation level test\n') % prints statement to command window
fprintf('\tRequired Percentage:%6.1f\t',req99)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act99)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
```

```
if act99 >= req99                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

num90   = length(find(verNorm(:,3) <= 1)); % 90% passing number
act90   = num90/numSamp*100;     % percent passing 90% level test
fprintf('%i of %i samples passed the 90%% ',num90,numSamp)
fprintf('validation level test\n') % prints statement to command window
fprintf('\tRequired Percentage:%6.1f\t',req90)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act90)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
if act90 >= req90                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

num50   = length(find(verNorm(:,4) > 1));  % 50% passing number
act50   = num50/numSamp*100;     % percent passing 50% level test
fprintf('%i of %i samples passed the 50%% ',num50,numSamp)
fprintf('validation level test\n') % prints statement to command window
fprintf('\tRequired Percentage:%6.1f\t',req50)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act50)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
if act50 >= req50                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

figure(1)                        % makes figure current
clf                              % clears current figure
hold on                          % turns hold on for current figure
plot(predRad(:,2),sampErr(:,2),'b.') % plots error vs predicted error
axisLimit   = ceil(max([predRad(:,2);sampErr(:,2)])); % max axis value
plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
axis equal                       % sets axes to equal scales
axis([0 axisLimit 0 axisLimit]) % sets axes limits
title('Vertical Error Samples vs. Predicted 99% Ellipsoid Errors')
xlabel('Predicted 99% Ellipsoid Error (m)') % adds label to x-axis
ylabel('Vertical Error (m)')     % adds label to y-axis
hold off                         % turns hold off for current figure
```

```
    figure(2)                       % makes figure current
    clf                             % clears current figure
    hold on                         % turns hold on for current figure
    plot(predRad(:,3),sampErr(:,3),'b.') % plots error vs predicted error
    axisLimit   = ceil(max([predRad(:,3);sampErr(:,3)])); % max axis value
    plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
    axis equal                      % sets axes to equal scales
    axis([0 axisLimit 0 axisLimit]) % sets axes limits
    title('Vertical Error Samples vs. Predicted 90% Ellipsoid Errors')
    xlabel('Predicted 90% Ellipsoid Error (m)') % adds label to x-axis
    ylabel('Vertical Error (m)')    % adds label to y-axis
    hold off                        % turns hold off for current figure

    figure(3)                       % makes figure current
    clf                             % clears current figure
    hold on                         % turns hold on for current figure
    plot(predRad(:,4),sampErr(:,4),'b.') % plots error vs predicted error
    axisLimit   = ceil(max([predRad(:,4);sampErr(:,4)])); % max axis value
    plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
    axis equal                      % sets axes to equal scales
    axis([0 axisLimit 0 axisLimit]) % sets axis limits
    title('Vertical Error Samples vs. Predicted 50% Ellipsoid Errors')
    xlabel('Predicted 50% Ellipsoid Error (m)') % adds label to x-axis
    ylabel('Vertical Error (m)')    % adds label to y-axis
    hold off                        % turns hold off for current figure

    if isempty(scalarCov)           % checks validation type
        figure(4)                   % makes figure current
        clf                         % clears current figure
        hold on                     % turns hold on for current figure
        axisLimit = ceil(max([predRad(:,2);sampErr(:,2)])); % max axis value
        plot([0 axisLimit/(predMag(1,2)/predMag(1,3))],...
             [0 axisLimit],'r-')    % plots 99% validation line
        plot([0 axisLimit],[0 axisLimit],'b-') % plots 90% validation line
        plot([0 axisLimit],...      % plots 50% validation line
             [0 (predMag(1,4)/predMag(1,3))*axisLimit],'g-')
        plot(predRad(:,3),sampErr(:,3),'b.') % plots error vs predicted error
        axis equal                  % sets axes to equal scales
        axis([0 axisLimit 0 axisLimit]) % sets axis limits
        title(['Vertical Error Samples vs. Predicted 90% Ellipsoid '...
               'Errors: 50%, 90%, 99% lines']) % adds title to figure
        xlabel('Predicted 90% Ellipsoid Error (m)') % adds label to x-axis
        ylabel('Vertical Error (m)') % adds label to y-axis
        legend('99% Validation Line','90% Validation Line',...
               '50% Validation Line','Location','northwest') % adds legend
        hold off                    % turns hold off for current figure
        drawnow                     % forces display to refresh
    end                             % ends loop checking validation type
end                                 % ends loop testing vertical
```

```
if size(errVal,2) == 2 || size(errVal,2) == 3 % checks if testing horizontal
    if size(errVal,2) == 2          % checks if testing 2d or 3d
        fprintf('\n\nThe horizontal (radial) component of the sample errors')
        fprintf(' will be analyzed (validated).\n\n') % prints statement
        horErr  = errVal;           % horizontal error values
        if isempty(predCov) == 0    % checks type of validation
            fprintf('\n\nThe horizontal (radial) component is being ')
            fprintf('analyzed (validated) with the predicted radials.\n')
            horCov  = predCov;      % horizontal error covariances
        elseif isempty(scalarCov) == 0 % checks type of validation
            fprintf('\n\nThe horizontal (radial) component is being ')
            fprintf('analyzed (validated) with the computed predicted ')
            fprintf('scalar accuracy metrics.\n') % prints statement
            horCov  = scalarCov;    % horizontal error covariance
        elseif isempty(scalarVal) == 0 % checks type of validation
            fprintf('\n\nThe horizontal (radial) component is being ')
            fprintf('analyzed (validated) with the entered predicted ')
            fprintf('scalar accuracy metrics.\n') % prints statement
            horCov  = scalarVal;    % CE90 values
        else                        % checks type of validation
            fprintf('\n\nAn error has occured in the determination of the ')
            fprintf('type of validation to apply to the horizontal ')
            fprintf('(radial) component.\n') % prints statement
            return                  % exits function
        end                         % ends loop checking validation type
    else                            % checks if testing 2d or 3d
        horErr  = errVal(:,1:2);    % horizontal error values
        if isempty(predCov) == 0    % checks type of validation
            fprintf('\n\nThe horizontal (radial) component is being ')
            fprintf('analyzed (validated) with the redicted radials.\n')
            horCov  = predCov(1:2,1:2,:); % horizontal error covariances
        elseif isempty(scalarCov) == 0 % checks type of validation
            fprintf('\n\nThe horizontal (radial) component is being ')
            fprintf('analyzed (validated) with the computed predicted ')
            fprintf('scalar accuracy metrics.\n')  % prints statement
            horCov  = scalarCov(1:2,1:2,:); % horizontal error covariance
        elseif isempty(scalarVal) == 0 % checks type of validation
            fprintf('\n\nThe horizontal (radial) component is being ')
            fprintf('analyzed (validated) with the entered predicted ')
            fprintf('scalar accuracy metrics.\n') % prints statement
            horCov  = scalarVal;    % CE90 values
        else                        % checks type of validation
            fprintf('\n\nAn error has occured in the determination of the ')
            fprintf('type of validation to apply to the horizontal ')
            fprintf('(radial) component.\n') % prints statement
            return                  % exits function
        end                         % ends loop checking type of validation
    end                             % ends loop checking tests to perform
```

```
    horDist = sqrt(horErr(:,1).^2+horErr(:,2).^2); % 2d radial distance
    sampErr = horDist*ones(1,4);     % sample radial errors
    if isempty(predCov) == 0         % checks validation type
        horNorm = zeros(numSamp,4); % creates matrix for normalized values
        predRad = zeros(numSamp,4); % creates matrix for predicted radials
        for n = 1:numSamp            % starts loop to cycle through samples
            %%% Computes normalized error
            horNorm(n,:) = sqrt(horErr(n,:)*inv(horCov(:,:,n))*...
                           transpose(horErr(n,:))).*predMag(2,:);
            predRad(n,:) = predMag(2,:)*horDist(n)*(horErr(n,:)*...
                           inv(horCov(:,:,n))*transpose(horErr(n,:))).^...
                           (-1/2);  % predicted sample radial error
        end                          % ends loop cycling through samples
    elseif isempty(scalarCov) == 0   % checks validation type
        prob    = [.99 .90 .50];     % probability levels for tests
        predRad = zeros(numSamp,4); % creates matrix for CE values
        fprintf('\tCalculating CE values for probability ')
        for m = 1:length(prob)       % starts loop to cycle thru probabilities
            fprintf('%i%% ... ',prob(m)*100) % prints statement
            for n = 1:numSamp        % starts loop to cycle through samples
                %%% computes CE value at specified probability
                predRad(n,m+1) = CEintegral(horCov(:,:,n),[0;0],prob(m));
            end                      % ends loop cycling through samples
        end                          % ends loop cycling thru probabilities
        fprintf('done.\n\n')         % prints statement to command window

        if strcmp(test999999,'on')   % checks if performing 99.9999% test
            predRad(:,1) = predMag(2,1)/predMag(2,3)*predRad(:,3); % CE999999
        end                          % ends loop checking 99.9999% test

        horNorm = sampErr./predRad; % horizontal normalized error
    elseif isempty(scalarVal) == 0   % checks validation type
%         predRad = scalarVal;         % predicted radial
%         sampErr = sqrt(horErr(:,1).^2+horErr(:,2).^2); % sample radial
errors
%         horNorm = sampErr./predRad; % horizontal normalized error

        predRad = zeros(numSamp,4); % creates matrix for predicted radials
        predRad(:,1) = (predMag(2,1)/predMag(2,3))*scalarVal; % fill matrix
        predRad(:,2) = (predMag(2,2)/predMag(2,3))*scalarVal; % fill matrix
        predRad(:,3) = scalarVal;    % fill matrix
        predRad(:,4) = (predMag(2,4)/predMag(2,3))*scalarVal; % fill matrix
        horNorm = sampErr./predRad; % horizontal normalized error
    end                              % ends loop checking validation type

    req9999 = 100;                   % sets pass percent for 99.9999% level
    %%% Interpolation for pass percentage from table
    if floor(levSc) == ceil(levSc)   % checks for value directly from table
        if strcmp(fidLev,'high')     % checks if using high fidelity
            req99   = HaccFid(1,floor(levSc)); % 99% level
            req90   = HaccFid(2,floor(levSc)); % 90% level
            req50   = HaccFid(3,floor(levSc)); % 50% level
```

```
    elseif strcmp(fidLev,'medium')  % checks if using medium fidelity
        req99   = HaccFid(4,floor(levSc)); % 99% level
        req90   = HaccFid(5,floor(levSc)); % 90% level
        req50   = HaccFid(6,floor(levSc)); % 50% level
    else                            % checks if using low fidelity
        req99   = HaccFid(7,floor(levSc)); % 99% level
        req90   = HaccFid(8,floor(levSc)); % 90% level
        req50   = HaccFid(9,floor(levSc)); % 50% level
    end                             % ends loop checking fidelity
else                                % checks for interpolated table value
    if strcmp(fidLev,'high')    % checks if using high fidelity
        req99   = HaccFid(1,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(1,floor(levSc))-...
                  HaccFid(1,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = HaccFid(2,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(2,floor(levSc))-...
                  HaccFid(2,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = HaccFid(3,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(3,floor(levSc))-...
                  HaccFid(3,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    elseif strcmp(fidLev,'medium')  % checks if using medium fidelity
        req99   = HaccFid(4,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(4,floor(levSc))-...
                  HaccFid(4,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = HaccFid(5,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(5,floor(levSc))-...
                  HaccFid(5,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = HaccFid(6,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(6,floor(levSc))-...
                  HaccFid(6,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    else                            % checks if using low fidelity
        req99   = HaccFid(7,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(7,floor(levSc))-...
                  HaccFid(7,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = HaccFid(8,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(8,floor(levSc))-...
                  HaccFid(8,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = HaccFid(9,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(HaccFid(9,floor(levSc))-...
                  HaccFid(9,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    end                             % ends loop checking fidelity
end                                 % ends loop checking table location
```

```
fprintf('FINAL VALIDATION RESULTS: The following results are for ')
fprintf('Horizontal (radial) Accuracy Validation.\n') % prints statement

if strcmp(test999999,'on')      % checks if preforming 99.9999% test
    num9999 = length(find(horNorm(:,1) <= 1)); % 99.9999% passing number
    act9999 = num9999/numSamp*100;  % percent passing 99.9999% level test
    fprintf('%i of %i samples passed the 99.9999%% ',num9999,numSamp)
    fprintf('validation level test\n') % prints statement
    fprintf('\tRequired Percentage:%6.1f\t',req9999) % prints statement
    fprintf('\tActual Percentage:%6.1f\t',act9999)   % prints statament
    fprintf('\tValidation Result: ') % prints statement to command window
    if act9999 >= req9999       % checks if test is passed
        fprintf('PASS\n')       % prints statement to command window
    else                        % checks if test is failed
        fprintf('FAIL\n')       % prints statement to command window
    end                         % ends loop checking test status
end                             % ends loop checking for 99.9999% test

num99   = length(find(horNorm(:,2) <= 1)); % 99% passing number
act99   = num99/numSamp*100;    % percent passing 99% level test
fprintf('%i of %i samples passed the 99%% ',num99,numSamp)
fprintf('validation level test\n') % prints statement
fprintf('\tRequired Percentage:%6.1f\t',req99)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act99)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
if act99 >= req99               % checks if test is passed
    fprintf('PASS\n')           % prints statement to command window
else                            % checks if test is failed
    fprintf('FAIL\n')           % prints statement to command window
end                             % ends loop checking test status

figure(5)                       % makes figure current
clf                             % clears current figure
hold on                         % turns hold on for current figure
plot(predRad(:,2),sampErr(:,2),'b.') % plots error vs predicted error
axisLimit = ceil(max([predRad(:,2);sampErr(:,2)])); % max axis value
plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
axis equal                      % sets axes to equal scales
axis([0 axisLimit 0 axisLimit]) % sets axes limits
title(['Horizontal (radial) Error Samples vs. Predicted 99% '...
       'Ellipsoid Errors'])     % adds title to figure
xlabel('Predicted 99% Ellipsoid Error (m)') % adds label to x-axis
ylabel('Horizontal (radial) Error (m)')     % adds label to y-axis
hold off                        % turns hold off for current figure

num90   = length(find(horNorm(:,3) <= 1)); % 90% passing number
act90   = num90/numSamp*100;    % percent passing 90% level test
fprintf('%i of %i samples passed the 90%% ',num90,numSamp)
fprintf('validation level test\n') % prints statement
fprintf('\tRequired Percentage:%6.1f\t',req90)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act90)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
```

```
if act90 >= req90                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

figure(6)                        % makes figure current
clf                              % clears current figure
hold on                          % turns hold on for current figure
plot(predRad(:,3),sampErr(:,3),'b.') % plots error vs predicted error
axisLimit = ceil(max([predRad(:,2);sampErr(:,2)])); % max axis value
plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
axis equal                       % sets axes to equal scales
axis([0 axisLimit 0 axisLimit]) % sets axes limits
title(['Horizontal (radial) Error Samples vs. Predicted 90% '...
       'Ellipsoid Errors'])      % adds title to figure
xlabel('Predicted 90% Ellipsoid Error (m)') % adds label to x-axis
ylabel('Horizontal (radial) Error (m)')     % adds label to y-axis
hold off                         % turns hold off for current figure

num50   = length(find(horNorm(:,4) > 1));  % 50% passing number
act50   = num50/numSamp*100;     % percent passing 50% level test
fprintf('%i of %i samples passed the 50%% ',num50,numSamp)
fprintf('validation level test\n') % prints statement
fprintf('\tRequired Percentage:%6.1f\t',req50)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act50)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
if act50 >= req50                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

figure(7)                        % makes figure current
clf                              % clears current figure
hold on                          % turns hold on for current figure
plot(predRad(:,4),sampErr(:,4),'b.') % plots error vs predicted error
axisLimit = ceil(max([predRad(:,4);sampErr(:,4)])); % max axis value
plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
axis equal                       % sets axes to equal scales
axis([0 axisLimit 0 axisLimit]) % sets axis limits
title(['Horizontal (radial) Error Samples vs. Predicted 50% '...
       'Ellipsoid Errors'])      % adds title to figure
xlabel('Predicted 50% Ellipsoid Error (m)') % adds label to x-axis
ylabel('Horizontal (radial) Error (m)')     % adds label to y-axis
hold off                         % turns hold off for current figure
```

```
    if isempty(predCov) == 0        % checks validation type
        figure(8)                   % makes figure current
        clf                         % clears current figure
        hold on                     % turns hold on for current figure
        axisLimit = ceil(max([predRad(:,3);sampErr(:,3)])); % max axis value
        plot([0 axisLimit/(predMag(2,2)/predMag(2,3))],...
             [0 axisLimit],'r-')    % plots 99% validation line
        plot([0 axisLimit],[0 axisLimit],'b-') % plots 90% validation line
        plot([0 axisLimit],...      % plots 50% validation line
             [0 (predMag(2,4)/predMag(2,3))*axisLimit],'g-')
        plot(predRad(:,3),sampErr(:,3),'b.') % plots error vs predicted error
        axis equal                  % sets axes to equal scales
        axis([0 axisLimit 0 axisLimit]) % sets axis limits
        title(['Horizontal (radial) Error Samples vs. Predicted 90% '...
               'Ellipsoid Errors: 50%, 90%, 99% lines']) % adds title
        xlabel('Predicted 90% Ellipsoid Error (m)') % adds label to x-axis
        ylabel('Horizontal (radial) Error (m)')     % adds label to y-axis
        legend('99% Validation Line','90% Validation Line',...
               '50% Validation Line','Location','northwest') % adds legend
        hold off                    % turns hold off for current figure
    end                             % ends loop checking validation type
    drawnow                         % forces display to refresh
end                                 % ends loop testing horizontal

if size(errVal,2) == 3              % checks if testing 3d radial
    radErr  = errVal;               % 3d radial error values
    if isempty(predCov) == 0        % checks type of validation
        fprintf('\n\nThe 3d radial component is being analyzed (validated) ')
        fprintf('with the predicted radials.\n') % prints statement
        radCov  = predCov;          % horizontal error covariances
    elseif isempty(scalarCov) == 0  % checks type of validation
        fprintf('\n\nThe 3d radial component is being analyzed (validated) ')
        fprintf('with the computed predicted scalar accuracy metrics.\n')
        radCov  = scalarCov;        % horizontal error covariance
    else                            % checks type of validation
        fprintf('\n\nAn error has occured in the determination of the ')
        fprintf('type of validation to apply to the 3d radial ')
        fprintf('component.\n')     % prints statement to command window
        return                      % exits function
    end                             % ends loop checking validation type

    radDist = sqrt(radErr(:,1).^2+radErr(:,2).^2+radErr(:,3).^2); % 3d dist
    sampErr = radDist*ones(1,4);    % sample radial errors
    if isempty(predCov) == 0        % checks validation type
        radNorm = zeros(numSamp,4); % creates matrix for normalized values
        predRad = zeros(numSamp,4); % creates matrix for predicted radials
        for n = 1:numSamp           % starts loop to cycle through samples
            %%% Computes normalized error
            radNorm(n,:) = sqrt(radErr(n,:)*inv(radCov(:,:,n))*...
                           transpose(radErr(n,:)))./predMag(3,:);
            predRad(n,:) = predMag(3,:)*radDist(n)*(radErr(n,:)*...
                           inv(radCov(:,:,n))*transpose(radErr(n,:))).^...
                           (-1/2);  % predicted sample radial error
        end                         % ends loop cycling through samples
```

```
elseif isempty(scalarCov) == 0  % checks validation type
    prob    = [.99 .90 .50];    % probability levels for tests
    predRad = zeros(numSamp,4); % creates matrix for SE values
    fprintf('\tCalculating SE values for probability ')
    for m = 1:length(prob)      % starts loop to cycle thru probabilities
        fprintf('%i%% ... ',prob(m)*100) % prints statement
        for n = 1:numSamp       % starts loop to cycle through samples
            %%% computes SE value at specified probability
            predRad(n,m+1) = SEintegral(radCov(:,:,n),[0;0;0],prob(m));
        end                     % ends loop cycling through samples
    end                         % ends loop cycling thru probabilities
    fprintf('done.\n\n')        % prints statement to command window

    if strcmp(test999999,'on')  % checks if performing 99.9999% test
        predRad(:,1) = predMag(3,1)/predMag(3,3)*predRad(:,3); % SE999999
    end                         % ends loop checking 99.9999% test

    radNorm = sampErr./predRad; % horizontal normalized error
end                             % ends loop checking validation type

req9999 = 100;                  % sets pass percent for 99.9999% level
%%% Interpolation for pass percentage from table
if floor(levSc) == ceil(levSc)  % checks if value directly in table
    if strcmp(fidLev,'high')    % checks if using high fidelity
        req99   = RaccFid(1,floor(levSc)); % 99% level
        req90   = RaccFid(2,floor(levSc)); % 90% level
        req50   = RaccFid(3,floor(levSc)); % 50% level
    elseif strcmp(fidLev,'medium')  % checks if using medium fidelity
        req99   = RaccFid(4,floor(levSc)); % 99% level
        req90   = RaccFid(5,floor(levSc)); % 90% level
        req50   = RaccFid(6,floor(levSc)); % 50% level
    else                                % checks if using low fidelity
        req99   = RaccFid(7,floor(levSc)); % 99% level
        req90   = RaccFid(8,floor(levSc)); % 90% level
        req50   = RaccFid(9,floor(levSc)); % 50% level
    end                         % ends loop checking fidelity
else                            % checks if value interpolated from table
    if strcmp(fidLev,'high')    % checks if using high fidelity
        req99   = RaccFid(1,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(1,floor(levSc))-...
                  RaccFid(1,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = RaccFid(2,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(2,floor(levSc))-...
                  RaccFid(2,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = RaccFid(3,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(3,floor(levSc))-...
                  RaccFid(3,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
```

```
    elseif strcmp(fidLev,'medium')  % checks if using medium fidelity
        req99   = RaccFid(4,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(4,floor(levSc))-...
                  RaccFid(4,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = RaccFid(5,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(5,floor(levSc))-...
                  RaccFid(5,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = RaccFid(6,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(6,floor(levSc))-...
                  RaccFid(6,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    else                          % checks if using low fidelity
        req99   = RaccFid(7,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(7,floor(levSc))-...
                  RaccFid(7,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 99% level
        req90   = RaccFid(8,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(8,floor(levSc))-...
                  RaccFid(8,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 90% level
        req50   = RaccFid(9,floor(levSc))-(sampLev(floor(levSc))-...
                  numSamp)*(RaccFid(9,floor(levSc))-...
                  RaccFid(9,ceil(levSc)))/(sampLev(floor(levSc))-...
                  sampLev(ceil(levSc))); % 50% level
    end                           % ends loop checking fidelity
end                               % ends loop checks table location

fprintf('FINAL VALIDATION RESULTS: The following results are for 3d ')
fprintf('radial Accuracy Validation.\n') % prints statement

if strcmp(test999999,'on')        % checks if performing 99.9999% test
    num9999 = length(find(radNorm(:,1) <= 1)); % 99.9999% passing number
    act9999 = num9999/numSamp*100;  % percent passing 99.9999% level test
    fprintf('%i of %i samples passed the 99.9999%% ',num9999,numSamp)
    fprintf('validation level test\n') % prints statement
    fprintf('\tRequired Percentage:%6.1f\t',req9999) % prints statement
    fprintf('\tActual Percentage:%6.1f\t',act9999)   % prints statament
    fprintf('\tValidation Result: ') % prints statement to command window
    if act9999 >= req9999         % checks if test is passed
        fprintf('PASS\n')         % prints statement to command window
    else                          % checks if test is failed
        fprintf('FAIL\n')         % prints statement to command window
    end                           % ends loop checking test status
end                               % ends loop checking for 99.9999% test

num99   = length(find(radNorm(:,2) <= 1)); % 99% passing number
act99   = num99/numSamp*100;      % percent passing 99% level test
fprintf('%i of %i samples passed the 99%% ',num99,numSamp)
fprintf('validation level test\n') % prints statement to command window
fprintf('\tRequired Percentage:%6.1f\t',req99)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act99)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
```

```
if act99 >= req99                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

num90   = length(find(radNorm(:,3) <= 1)); % 90% passing number
act90   = num90/numSamp*100;     % percent passing 90% level test
fprintf('%i of %i samples passed the 90%% ',num90,numSamp)
fprintf('validation level test\n') % prints statement to command window
fprintf('\tRequired Percentage:%6.1f\t',req90)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act90)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
if act90 >= req90                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

num50   = length(find(radNorm(:,4) > 1));  % 50% passing number
act50   = num50/numSamp*100;     % percent passing 50% level test
fprintf('%i of %i samples passed the 50%% ',num50,numSamp)
fprintf('validation level test\n') % prints statement to command window
fprintf('\tRequired Percentage:%6.1f\t',req50)  % prints statement
fprintf('\tActual Percentage:%6.1f\t',act50)    % prints statement
fprintf('\tValidation Result: ') % prints statement to command window
if act50 >= req50                % checks if test is passed
    fprintf('PASS\n')            % prints statement to command window
else                             % checks if test is failed
    fprintf('FAIL\n')            % prints statement to command window
end                              % ends loop checking test status

figure(9)                        % makes figure current
clf                              % clears current figure
hold on                          % turns hold on for current figure
plot(predRad(:,2),sampErr(:,2),'b.') % plots error vs predicted error
axisLimit   = ceil(max([predRad(:,2);sampErr(:,2)])); % max axis value
plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
axis equal                       % sets axes to equal scales
axis([0 axisLimit 0 axisLimit]) % sets axes limits
title(['3d Radial Error Samples vs. Predicted 99% '...
       'Ellipsoid Errors'])      % adds title to figure
xlabel('Predicted 99% Ellipsoid Error (m)') % adds label to x-axis
ylabel('3d Radial Error (m)')    % adds label to y-axis
hold off                         % turns hold off for current figure
```

```
    figure(10)                          % makes figure current
    clf                                 % clears current figure
    hold on                             % turns hold on for current figure
    plot(predRad(:,3),sampErr(:,3),'b.') % plots error vs predicted error
    axisLimit   = ceil(max([predRad(:,3);sampErr(:,3)])); % max axis value
    plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
    axis equal                          % sets axes to equal scales
    axis([0 axisLimit 0 axisLimit]) % sets axes limits
    title(['3d Radial Error Samples vs. Predicted 90% '...
           'Ellipsoid Errors'])         % adds title to figure
    xlabel('Predicted 90% Ellipsoid Error (m)') % adds label to x-axis
    ylabel('3d Radial Error (m)')       % adds label to y-axis
    hold off                            % turns hold off for current figure

    figure(11)                          % makes figure current
    clf                                 % clears current figure
    hold on                             % turns hold on for current figure
    plot(predRad(:,4),sampErr(:,4),'b.') % plots error vs predicted error
    axisLimit   = ceil(max([predRad(:,4);sampErr(:,4)])); % max axis value
    plot([0 axisLimit],[0 axisLimit],'b-') % plots validation line
    axis equal                          % sets axes to equal scales
    axis([0 axisLimit 0 axisLimit]) % sets axis limits
    title(['3d Radial Error Samples vs. Predicted 50% '...
           'Ellipsoid Errors'])         % adds title to figure
    xlabel('Predicted 50% Ellipsoid Error (m)') % adds label to x-axis
    ylabel('3d Radial Error (m)')       % adds label to y-axis
    hold off                            % turns hold off for current figure

    if isempty(scalarCov)               % checks validation type
        figure(12)                      % makes figure current
        clf                             % clears current figure
        hold on                         % turns hold on for current figure
        axisLimit = ceil(max([predRad(:,3);sampErr(:,3)])); % max axis value
        plot([0 axisLimit/(predMag(3,2)/predMag(3,3))],...
             [0 axisLimit],'r-')        % plots 99% validation line
        plot([0 axisLimit],[0 axisLimit],'b-') % plots 90% validation line
        plot([0 axisLimit],...          % plots 50% validation line
             [0 (predMag(3,4)/predMag(3,3))*axisLimit],'g-')
        plot(predRad(:,3),sampErr(:,3),'b.') % plots error vs predicted error
        axis equal                      % sets axes to equal scales
        axis([0 axisLimit 0 axisLimit]) % sets axis limits
        title(['3d Radial Error Samples vs. Predicted 90% Ellipsoid '...
               'Errors: 50%, 90%, 99% lines']) % adds title to figure
        xlabel('Predicted 90% Ellipsoid Error (m)') % adds label to x-axis
        ylabel('3d Radial Error (m)') % adds label to y-axis
        legend('99% Validation Line','90% Validation Line',...
               '50% Validation Line','Location','northwest') % adds legend
        hold off                        % turns hold off for current figure
        drawnow                         % forces display to refresh
    end                                 % ends loop checking validation type
end                                     % ends loop testing 3d radial
```

C.2 Examples

All examples use the same process to generate covariances and the corresponding error samples. This process involves random number generation to make each covariance, error sample, and example different. This code is for the sole purpose of generating the necessary values for the following examples, and is not part of the Predicted Accuracy Validation function.

The MATLAB code for this generation is as follows.

```
%%% Specify 2 different "basic" actual (true) 3x3 error covariance matrices
%%% for "variability and modeling within "operational constraints"; values
%%% essentially arbitrary but must correspond to valid covariance matrices
%%% (symmetric and positive definite) for 3d geolocation errors:
P1      = zeros(3,3);               % creates first covariance
P1(1,1) = 1.2;                      % fills part of covariance
P1(2,2) = 1.1;                      % fills part of covariance
P1(3,3) = 1.3;                      % fills part of covariance
P1(1,2) = .2*sqrt(P1(1,1)*P1(2,2)); % fills part of covariance
P1(2,1) = P1(1,2);                  % fills part of covariance
P1(1,3) = .1*sqrt(P1(1,1)*P1(3,3)); % fills part of covariance
P1(3,1) = P1(1,3);                  % fills part of covariance
P1(2,3) = .8*sqrt(P1(2,2)*P1(3,3)); % fills part of covariance
P1(3,2) = P1(2,3);                  % fills part of covariance
P1      = 3*P1;                     % scales covariance

P2      = zeros(3,3);               % creates second covariance
P2(1,1) = 2.2;                      % fills part of covariance
P2(2,2) = 1.6;                      % fills part of covariance
P2(3,3) = 1.8;                      % fills part of covariance
P2(1,2) = .2*sqrt(P2(1,1)*P2(2,2)); % fills part of covariance
P2(2,1) = P2(1,2);                  % fills part of covariance
P2(1,3) = .1*sqrt(P2(1,1)*P2(3,3)); % fills part of covariance
P2(3,1) = P2(1,3);                  % fills part of covariance
P2(2,3) = .8*sqrt(P2(2,2)*P2(3,3)); % fills part of covariance
P2(3,2) = P2(2,3);                  % fills part of covariance
P2      = 3*P2;                     % scales covariance

%%% Loop over number of samples to create "realizations"; perturb each
%%% sample and covariance so slightly differnt
fprintf('Starting creation of sample value: ... ') % prints statement
P   = zeros(3,3,numSamp);           % creates covariance matrices
X   = zeros(numSamp,3);             % creates matrix for sample values
for n = 1:numSamp                   % creates loop to cycle through samples
    if mod(n,10) == 0               % checks current loop counter
        fprintf('%i ... ',n)        % prints statement to command window
    end                             % ends loop cycling through samples

    if mod(n,2) == 0                % checks current loop counter
        Puse    = P1;               % sets current sample covariance
    else                            % checks current loop counter
        Puse    = P2;               % sets current sample covariance
    end                             % ends loop checking current loop counter
```

```
    Puse    = Puse+diag(.01*diag(Puse).*randn(3,1)); % perturbs covariance

    X(n,:)   = sqrtm(Puse)*randn(3,1); % generates sample values
    P(:,:,n) = .95*Puse;               % predicted error covariance matrix
end                                    % ends loop cycling through samples
fprintf('done\n')                      % prints statement to command window
```

C.2.1 Example 1: Radial Method of Calculating the Normalized Error

Example 1: This example uses the radial (ellipsoidal based) method of calculating the normalized error. This will be done for a group of (100) 3d inputs. The function call and the results follow.

The error samples values for this call are in the variable *X*, and would take the form of an *n*-by-3 matrix, where *n* is the number of error samples. Here the format is shown with five error samples, but this example used 100.

$$X = \begin{bmatrix} 2.18 & -0.50 & -2.31 \\ 5.15 & 0.86 & 4.59 \\ -0.44 & 1.21 & 2.99 \\ -2.07 & 2.31 & 3.45 \\ -0.77 & -0.29 & -1.39 \end{bmatrix}$$

The predicted covariances for this call are in the variable *P*. The covariances are stored in a three dimensional matrix being 3-by-3-by-*n*, where *n* is the number of error samples. The format is shown again with only five covariances, but this example used 100.

Covariance corresponding to first error sample (P matrix Layer 1):

$$P_{(3x3x1)} = \begin{bmatrix} 6.3037 & 1.0694 & 0.5671 \\ 1.0694 & 4.6436 & 3.8693 \\ 0.5671 & 3.8693 & 5.0141 \end{bmatrix}$$

Covariance corresponding to second error sample (P matrix Layer 2):

$$P_{(3x3x1)} = \begin{bmatrix} 3.4052 & 0.6549 & 0.3560 \\ 0.6549 & 3.1457 & 2.7265 \\ 0.3560 & 2.7265 & 3.8376 \end{bmatrix}$$

Covariance corresponding to third error sample (P matrix Layer 3):

$$P_{(3x3x1)} = \begin{bmatrix} 6.3155 & 1.0694 & 0.5671 \\ 1.0694 & 4.5571 & 3.8693 \\ 0.5671 & 3.8693 & 5.1667 \end{bmatrix}$$

Covariance corresponding to fourth error sample (P matrix Layer 4):

$$P_{(3x3x1)}= \begin{bmatrix} 3.4682 & 0.6549 & 0.3560 \\ 0.6549 & 3.1794 & 2.7265 \\ 0.3560 & 2.7265 & 3.7299 \end{bmatrix}$$

Covariance corresponding to fifth error sample (P matrix Layer 5):

$$P_{(3x3x1)}= \begin{bmatrix} 6.3007 & 1.0694 & 0.5671 \\ 1.0694 & 4.6072 & 3.8693 \\ 0.5671 & 3.8693 & 5.1673 \end{bmatrix}$$

Function Call:

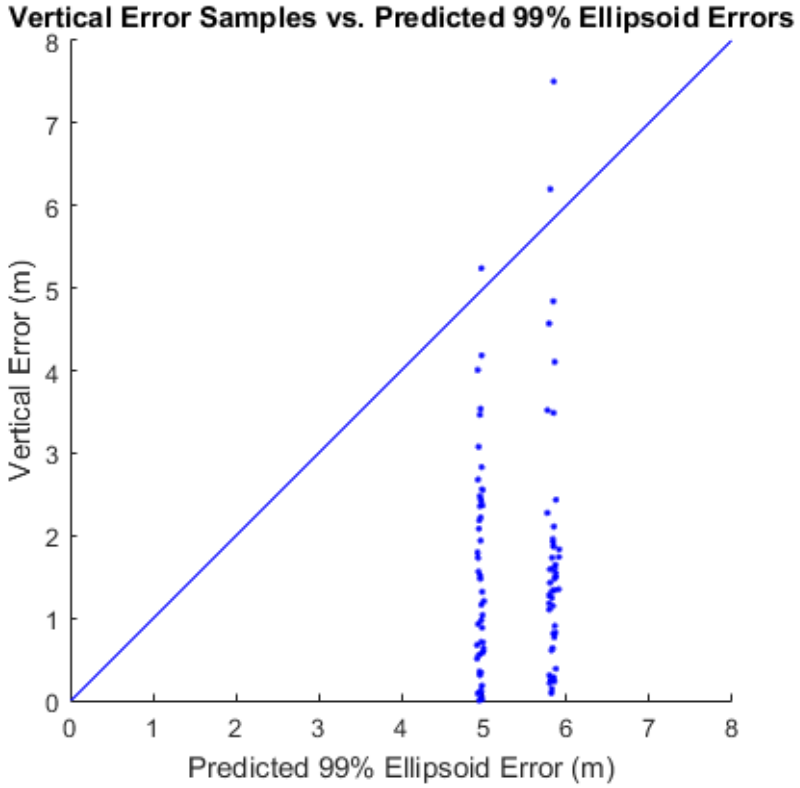
```
PredictedAccuracyValidation(X,'PredictedCovariance',P,'MinSamples',75)
```

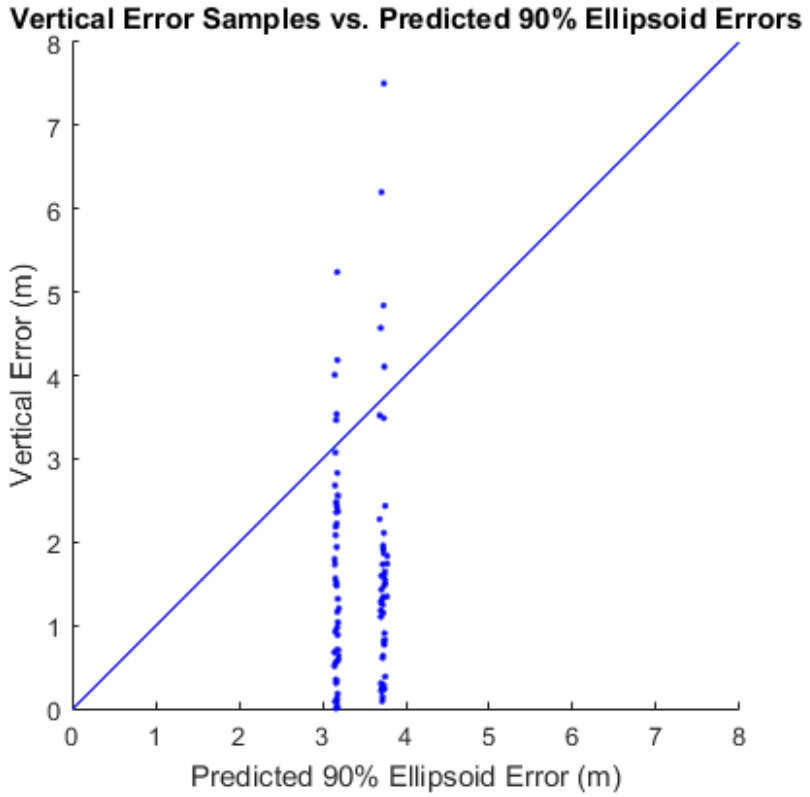
Function Results:

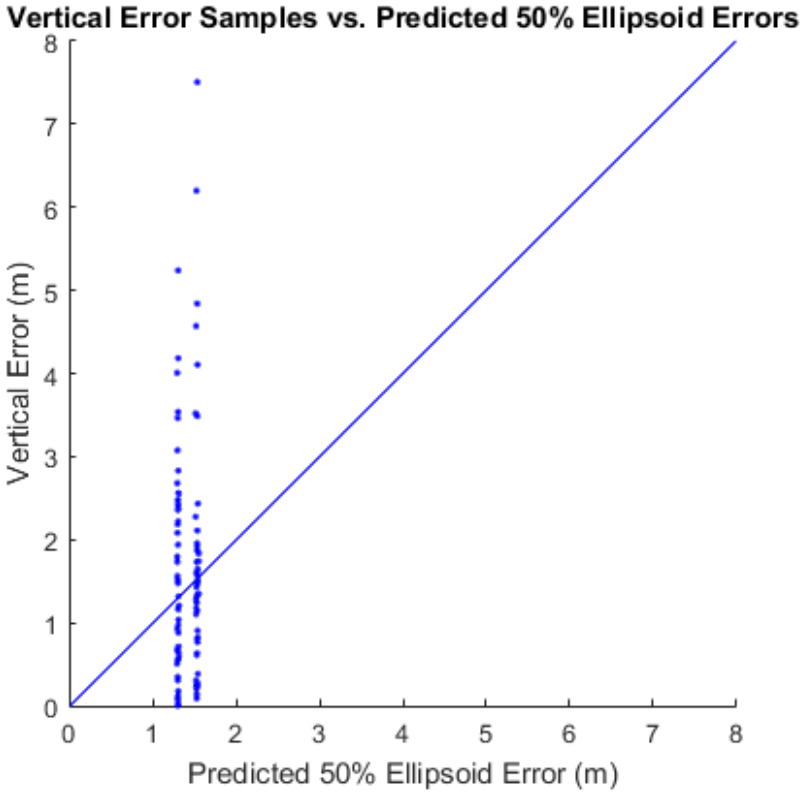
The current Predicted Accuracy Validation activity is validating 100 samples.
The validation will be performed using the predicted radials.
The Fidelity Level being used is high (default).
The number of error samples (100) meets the minimum number of i.i.d. error samples (75).

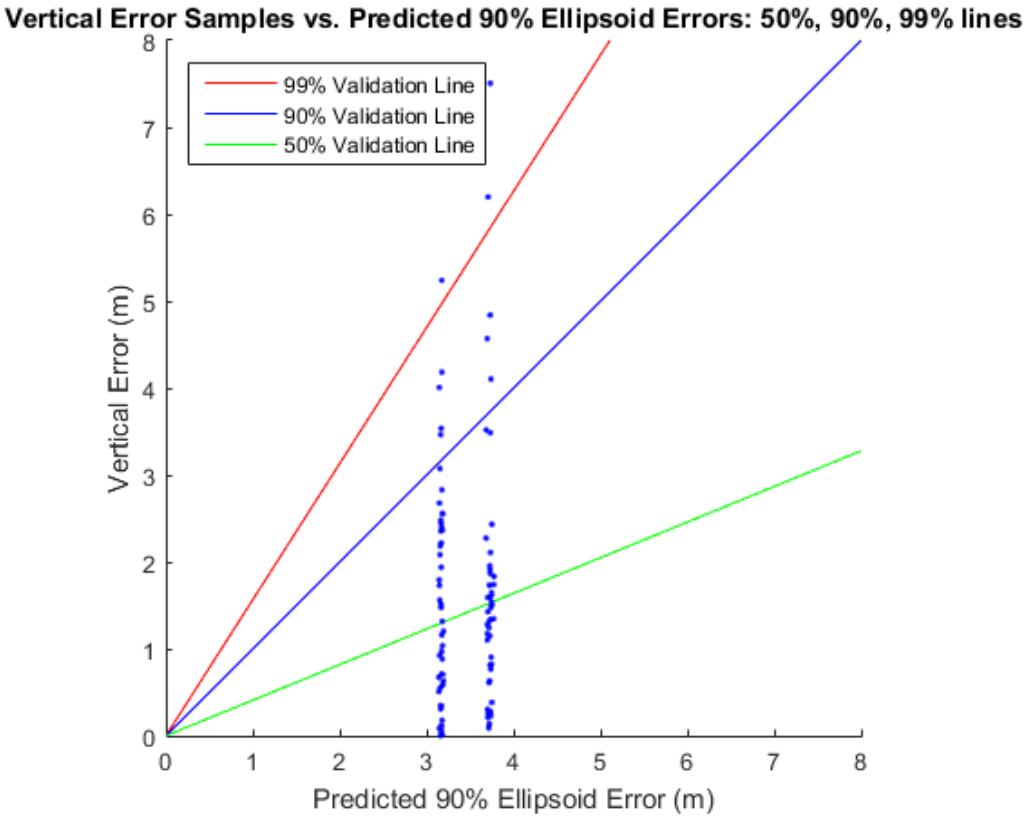
The vertical, horizontal (radial), and 3d radial components of the sample errors will be analyzed (validated).

The vertical component is being analyzed (validated) with the predicted radials.
FINAL VALIDATION RESULTS: The following results are for Vertical Accuracy Validation.
97 of 100 samples passed the 99% validation level test
Required Percentage: 95.0 Actual Percentage: 97.0 Validation Result: PASS
90 of 100 samples passed the 90% validation level test
Required Percentage: 83.0 Actual Percentage: 90.0 Validation Result: PASS
48 of 100 samples passed the 50% validation level test
Required Percentage: 40.0 Actual Percentage: 48.0 Validation Result: PASS

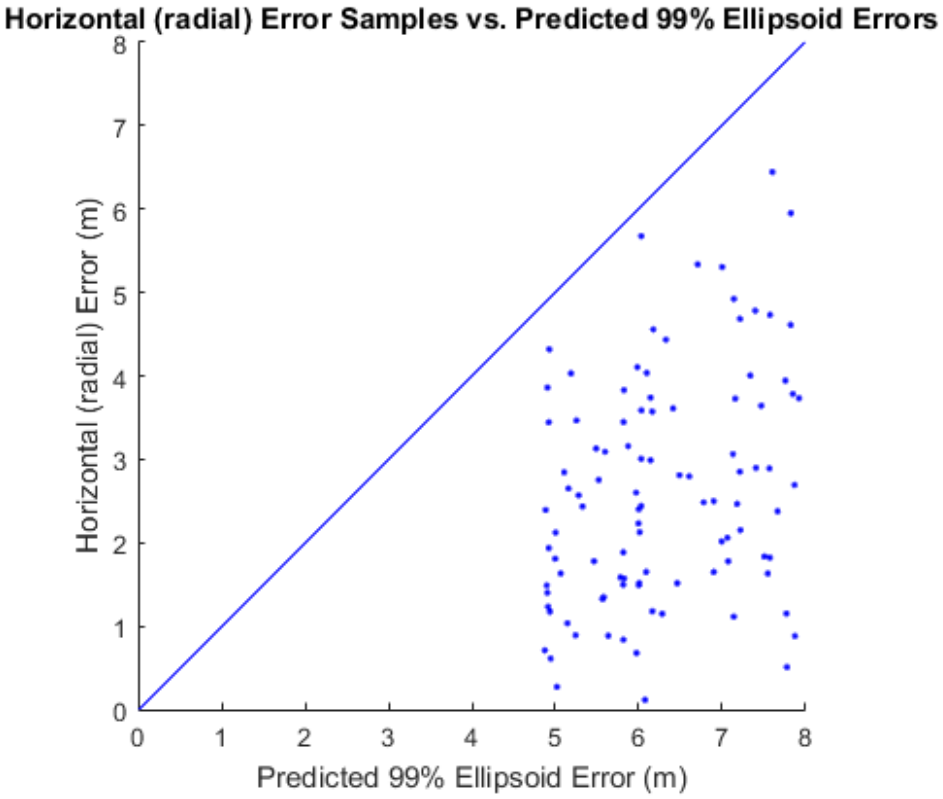

Vertical Error Samples vs. Predicted 99% Ellipsoid Errors
Vertical Error (m)
Predicted 99% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8

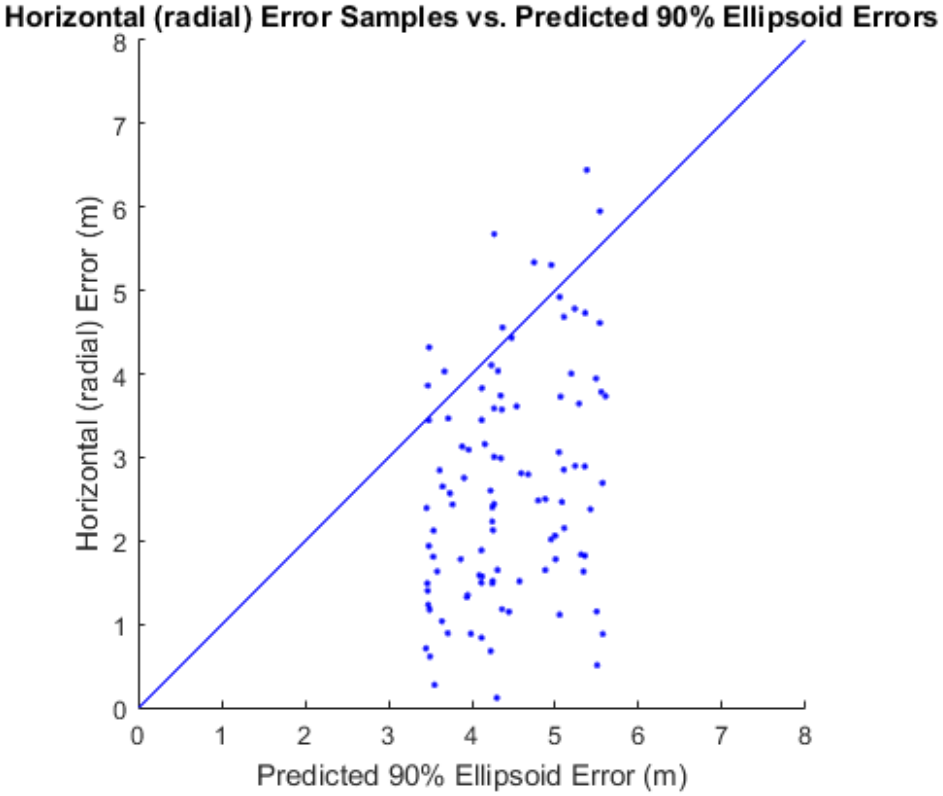

Vertical Error Samples vs. Predicted 90% Ellipsoid Errors
Vertical Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8

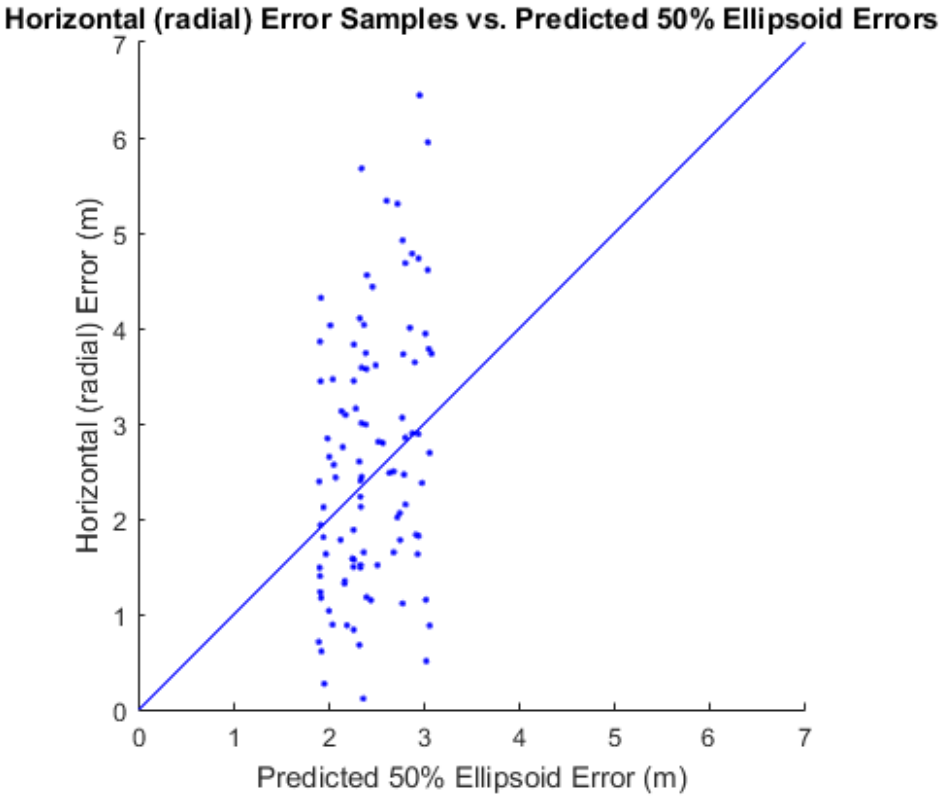

Vertical Error Samples vs. Predicted 50% Ellipsoid Errors
Vertical Error (m)
Predicted 50% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8

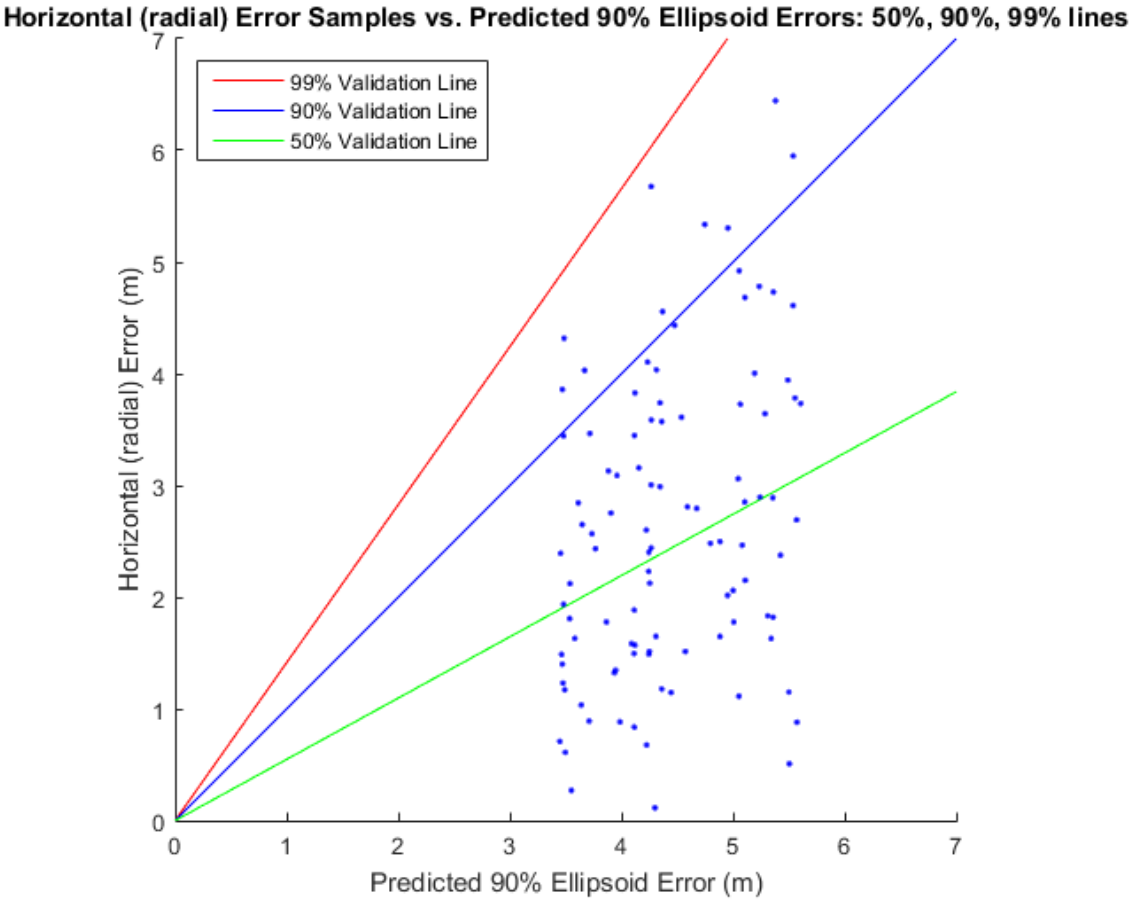

Vertical Error Samples vs. Predicted 90% Ellipsoid Errors: 50%, 90%, 99% lines
99% Validation Line
90% Validation Line
50% Validation Line
Vertical Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8

The horizontal (radial) component is being analyzed (validated) with the predicted radials.
FINAL VALIDATION RESULTS: The following results are for Horizontal (radial) Accuracy Validation.
100 of 100 samples passed the 99% validation level test
Required Percentage: 95.0 Actual Percentage: 100.0 Validation Result: PASS
91 of 100 samples passed the 90% validation level test
Required Percentage: 83.0 Actual Percentage: 91.0 Validation Result: PASS
52 of 100 samples passed the 50% validation level test
Required Percentage: 39.0 Actual Percentage: 52.0 Validation Result: PASS

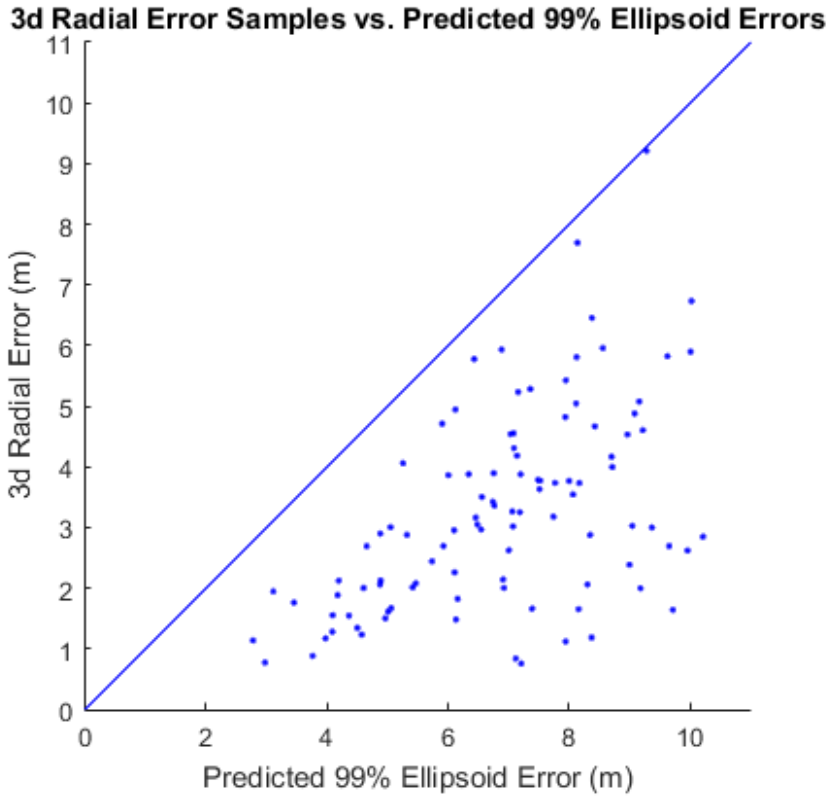


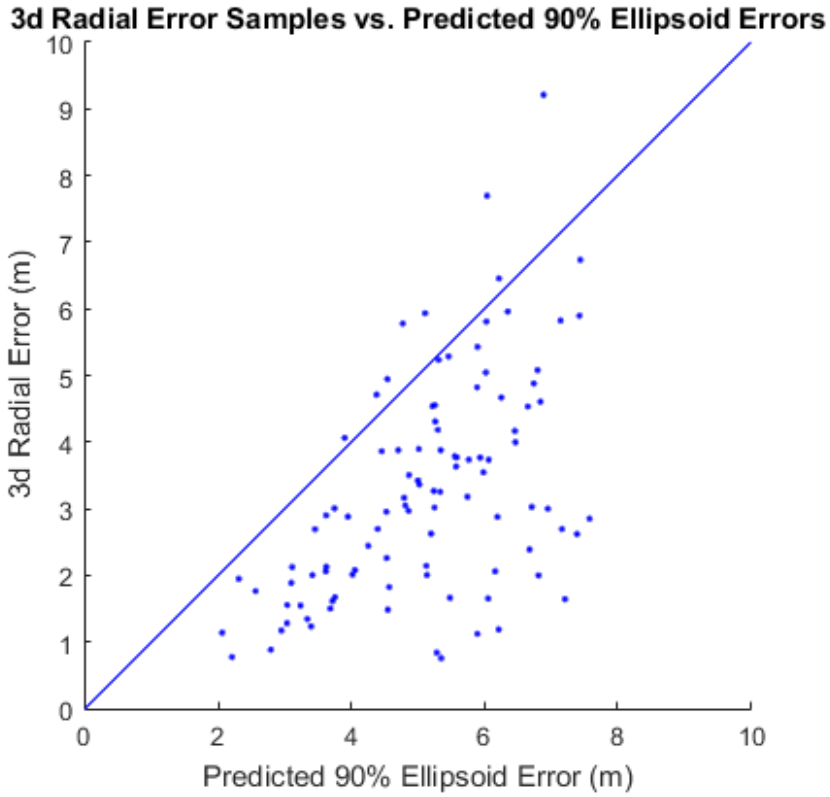

Horizontal (radial) Error Samples vs. Predicted 90% Ellipsoid Errors
Horizontal (radial) Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8

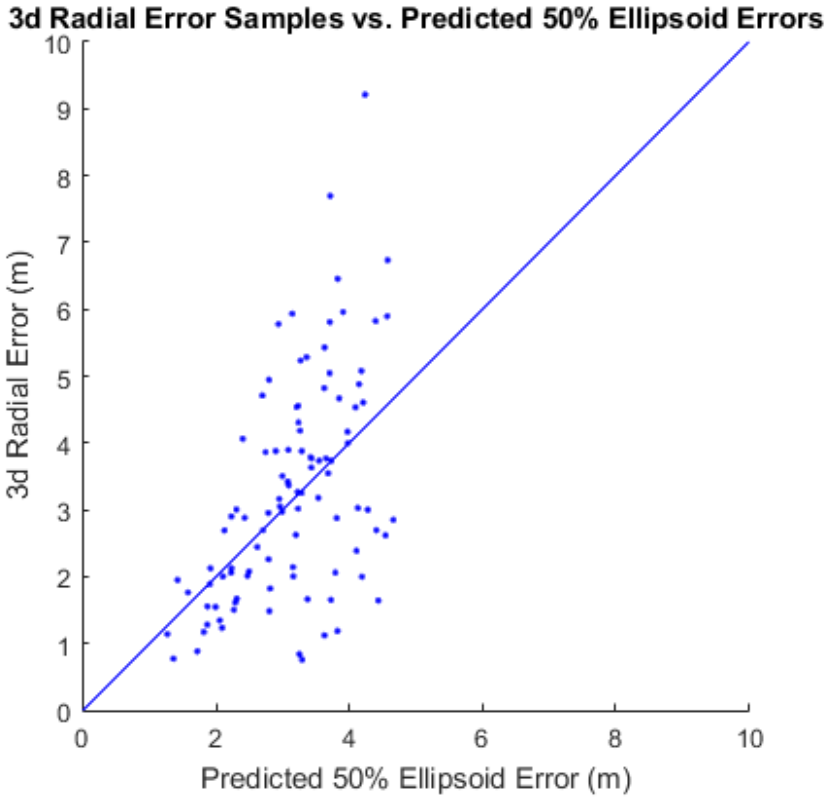

Horizontal (radial) Error Samples vs. Predicted 50% Ellipsoid Errors
Horizontal (radial) Error (m)
Predicted 50% Ellipsoid Error (m)
0
1
2
3
4
5
6
7

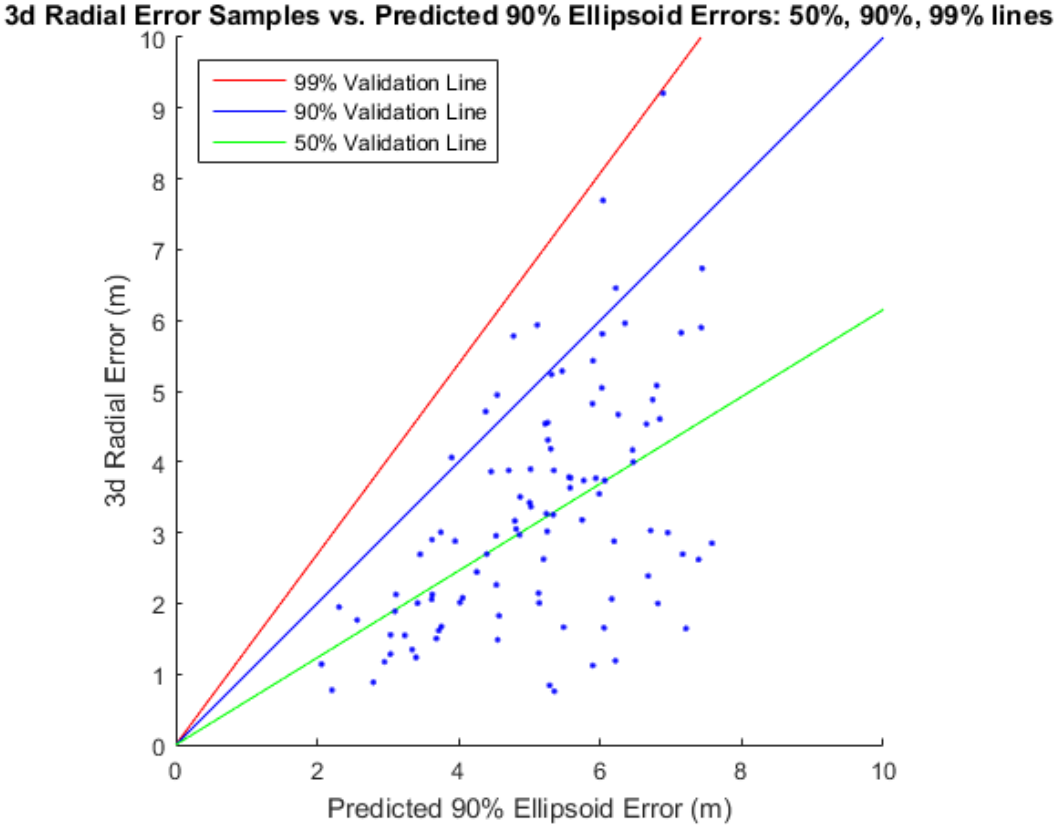


The 3d radial component is being analyzed (validated) with the predicted radials.
FINAL VALIDATION RESULTS: The following results are for 3d radial Accuracy Validation.
100 of 100 samples passed the 99% validation level test
Required Percentage: 94.0 Actual Percentage: 100.0 Validation Result: PASS
92 of 100 samples passed the 90% validation level test
Required Percentage: 82.0 Actual Percentage: 92.0 Validation Result: PASS
53 of 100 samples passed the 50% validation level test
Required Percentage: 37.0 Actual Percentage: 53.0 Validation Result: PASS


3d Radial Error Samples vs. Predicted 99% Ellipsoid Errors
3d Radial Error (m)
Predicted 99% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8
9
10
11
0
2
4
6
8
10


3d Radial Error Samples vs. Predicted 90% Ellipsoid Errors
3d Radial Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10


3d Radial Error Samples vs. Predicted 50% Ellipsoid Errors
3d Radial Error (m)
Predicted 50% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8
9
10


3d Radial Error Samples vs. Predicted 90% Ellipsoid Errors: 50%, 90%, 99% lines
99% Validation Line
90% Validation Line
50% Validation Line
3d Radial Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8
9
10

C.2.2 Example 2: Computed Scalar Accuracy Method of Calculating the Normalized Error

Example 2: This example uses the computed predicted scalar accuracy method of calculating the normalized error. This will be done for a group of 1d inputs. (Note: for ease of experiment, the predicted covariance values actually correspond to 'high' fidelity although 'medium' is specified in the function call.) The function call and the results follow.

The error samples values for this call are in the variable *X*, and would take the form of an *n*-by-1 vector, where *n* is the number of error samples. Here the format is shown with five error samples, but this example used 100.

$$X = \begin{bmatrix} -2.31 \\ 4.59 \\ 2.99 \\ 3.45 \\ -1.39 \end{bmatrix}$$

The predicted variances for this call are in the variable *P*. The variances are stored in a vector being *n*-by-1, where *n* is the number of error samples. The format is shown again with only five variances, but this example used 100.

$$P = \begin{bmatrix} 5.0141 \\ 3.8376 \\ 5.1667 \\ 3.7299 \\ 5.1673 \end{bmatrix}$$

Function Call:

```
PredictedAccuracyValidation(X,'ComputedScalar',P,
                            'FidelityLevel','medium',
                            'Test0.999999','on')
```

Function Results:

The current Predicted Accuracy Validation activity is validating 100 samples.
The validation will be performed using the predicted scalar accuracy metrics computed from error covariance matrices.
The Fidelity Level being used is medium.

The vertical component of the sample errors will be analyzed (validated).

The vertical component is being analyzed (validated) with the computed predicted scalar accuracy metrics.

Calculating LE values for probability 99% ... 90% ... 50% ... done.

FINAL VALIDATION RESULTS: The following results are for Vertical Accuracy Validation.

100 of 100 samples passed the 99.9999% validation level test

Required Percentage: 100.0 Actual Percentage: 100.0 Validation Result: PASS

99 of 100 samples passed the 99% validation level test

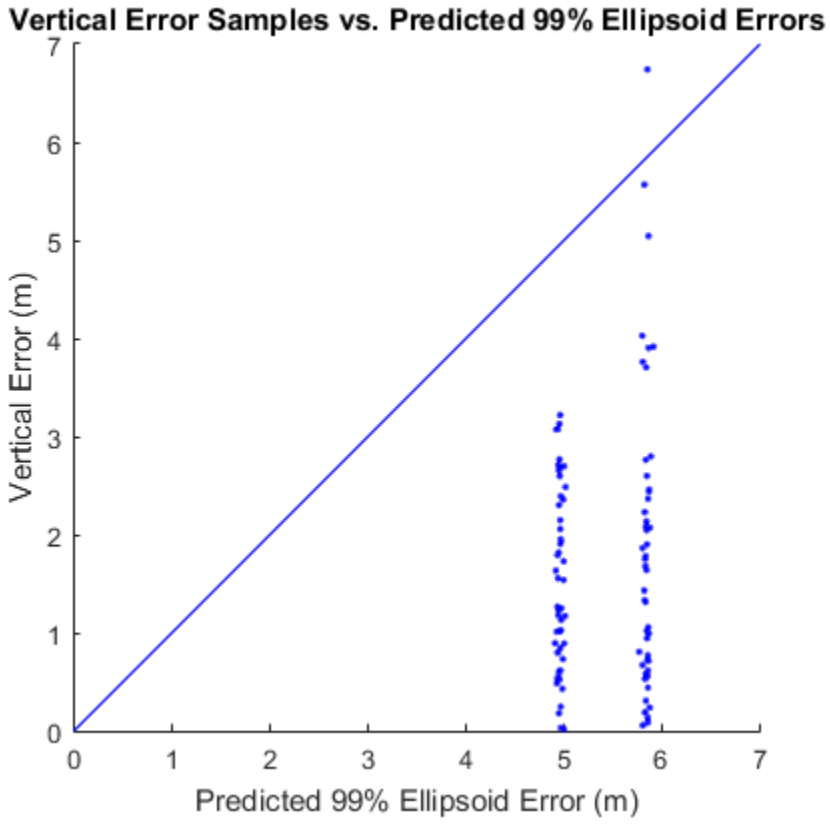
Required Percentage: 91.0 Actual Percentage: 99.0 Validation Result: PASS

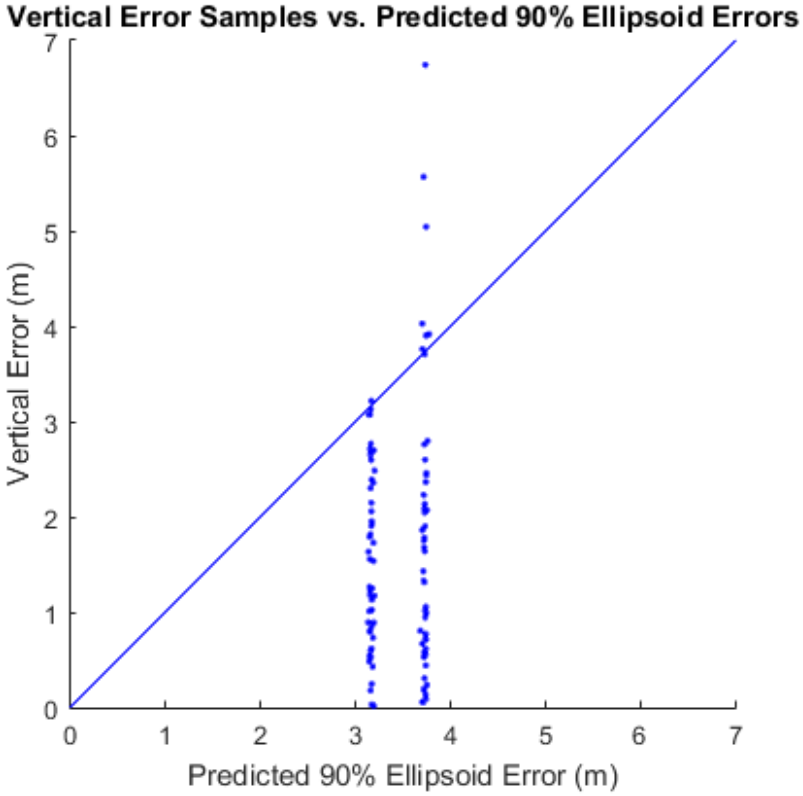
92 of 100 samples passed the 90% validation level test

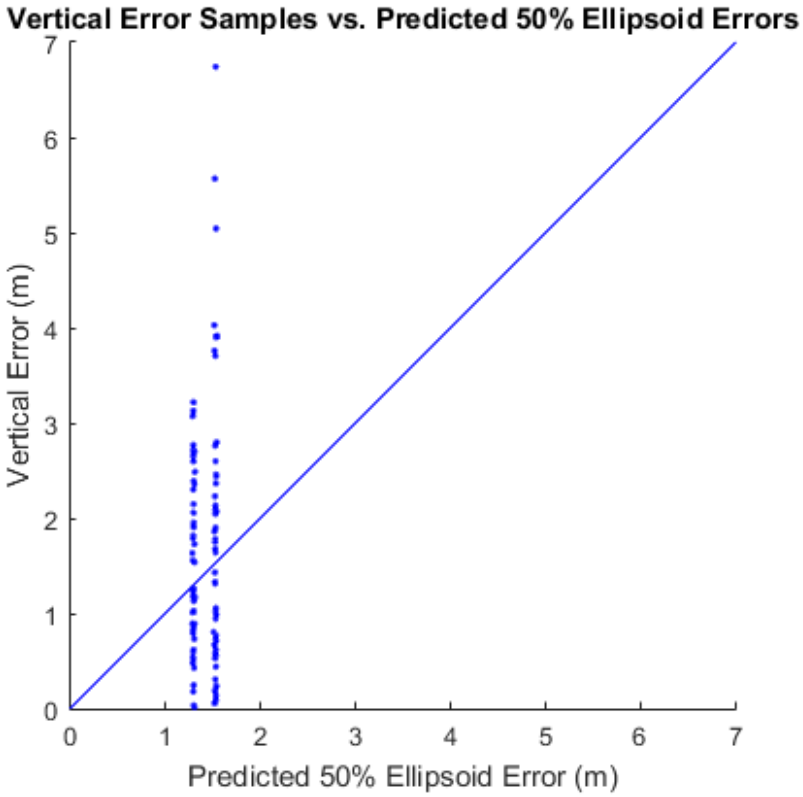
Required Percentage: 78.0 Actual Percentage: 92.0 Validation Result: PASS

50 of 100 samples passed the 50% validation level test

Required Percentage: 34.0 Actual Percentage: 50.0 Validation Result: PASS




Vertical Error Samples vs. Predicted 90% Ellipsoid Errors
Vertical Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7


Vertical Error Samples vs. Predicted 50% Ellipsoid Errors
Vertical Error (m)
Predicted 50% Ellipsoid Error (m)
0
1
2
3
4
5
6
7

C.2.3 Example 3: Entered Predicted Scalar Accuracy Method of Calculating the Normalized Error

Example 3: This example uses the entered predicted scalar accuracy method of calculating the normalized error. This will be done for a group of 2d inputs (this option only works with 2d inputs and corresponding CE90 values). The function call and the results follow.

The error samples values for this call are in the variable *X*, and would take the form of an *n*-by-2 matrix, where *n* is the number of error samples. Here the format is shown with five error samples, but this example used 100.

$$X = \begin{bmatrix} 2.18 & -0.50 \\ 5.15 & 0.86 \\ -0.44 & 1.21 \\ -2.07 & 2.31 \\ -0.77 & -0.29 \end{bmatrix}$$

This example uses the option to use input CE90 values. Internal to the predicted accuracy validation function, the corresponding CE99 and CE50 values are approximated by scaling the CE90 values. This option is available because some users will not have access to the covariance matrices corresponding to the error samples, but they will have the CE90 value calculated from those covariance matrices. If the user has the corresponding covariance matrices instead, then the method used in Example 1 is recommended. If the corresponding covariance matrices are available and the use of the LE, CE, and SE values are desired instead, then the method used in Example 2 is applicable.

For this current example the CE90 values were calculated using the function, CEintegral, found in "Accuracy and Predicted Accuracy in the NSG: Predictive Statistics; Technical Guidance Document TGD 2a Section 7: Reference Appendix C.3 Pseudo-code". The CE90 calculation is done from a 2-by-2 covariance matrix *P*, and the call can be seen below. This call is not part of the Predicted Accuracy Validation function, but used to obtain the desired input. Repeating this call for the number of error sample covariances and placing the resulting CE90 values in an *n*-by-1 vector produces the needed input for the function call of this example. Again, *n* is the number of error samples. The format is shown again with only five CE90 values, but this example used 100.

```
CE90 = CEintegral(P,[0;0],.90);
```

Where a sample P is:

$$P = \begin{bmatrix} 6.3037 & 1.0694 \\ 1.0694 & 4.6436 \end{bmatrix}$$

$$CE90 = \begin{bmatrix} 5.03 \\ 3.89 \\ 5.02 \\ 3.92 \\ 5.03 \end{bmatrix}$$

Function Call:

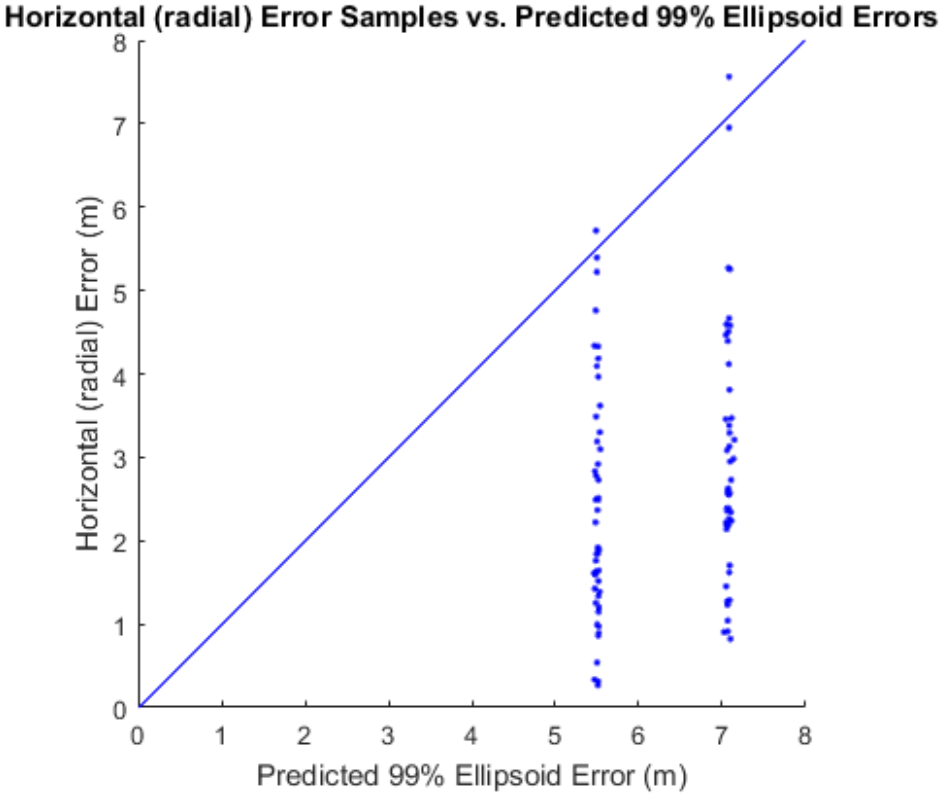
```
PredictedAccuracyValidation(X,'EnteredScalar',CE90)
```

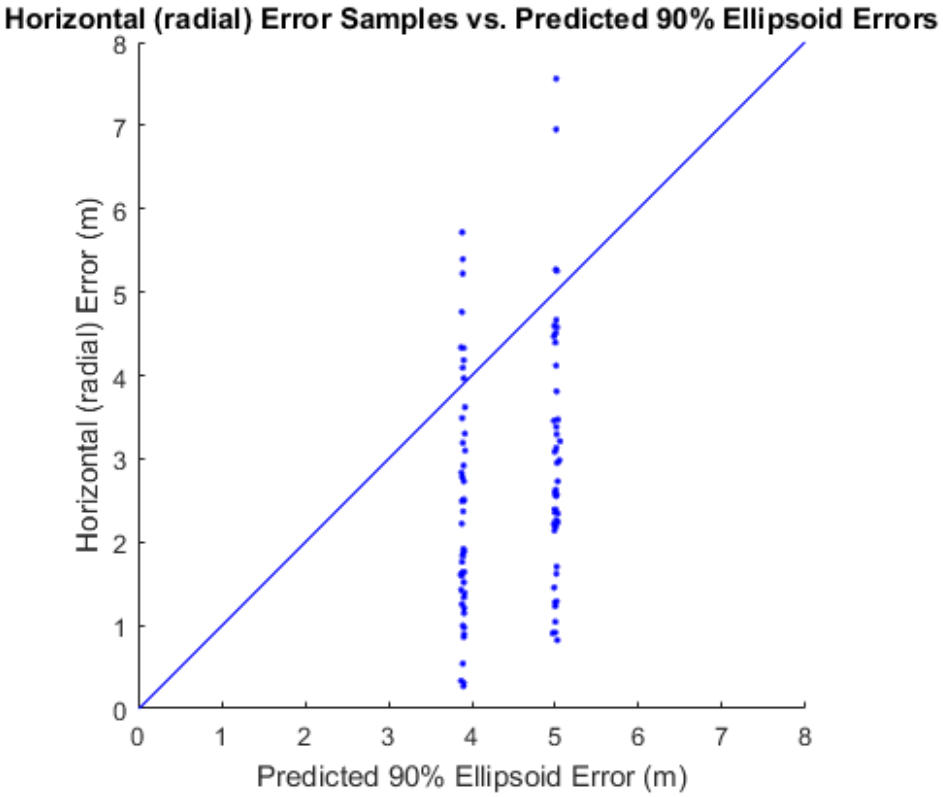
Function Results:

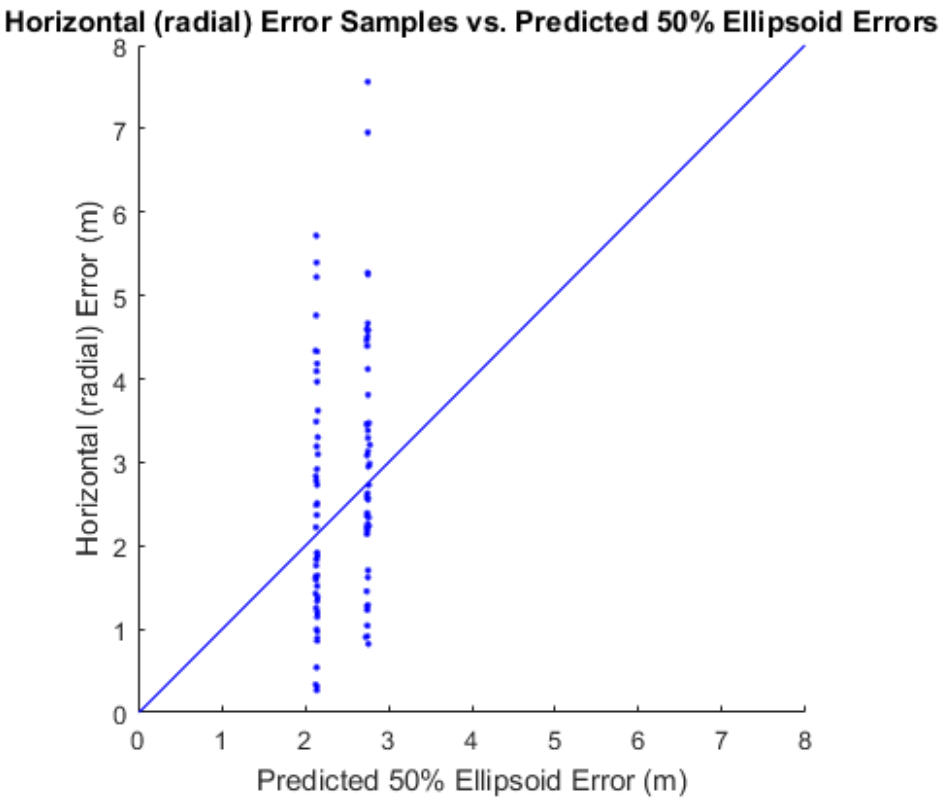
The current Predicted Accuracy Validation activity is validating 100 samples.
The validation will be performed using the predicted scalar accuracy metrics entered by the user.
The Fidelity Level being used is high (default).

The horizontal (radial) component of the sample errors will be analyzed (validated).

The horizontal (radial) component is being analyzed (validated) with the entered predicted scalar accuracy metrics.
FINAL VALIDATION RESULTS: The following results are for Horizontal (radial) Accuracy Validation.
98 of 100 samples passed the 99% validation level test
Required Percentage: 95.0 Actual Percentage: 98.0 Validation Result: PASS
87 of 100 samples passed the 90% validation level test
Required Percentage: 83.0 Actual Percentage: 87.0 Validation Result: PASS
45 of 100 samples passed the 50% validation level test
Required Percentage: 39.0 Actual Percentage: 45.0 Validation Result: PASS


Horizontal (radial) Error Samples vs. Predicted 99% Ellipsoid Errors
Horizontal (radial) Error (m)
Predicted 99% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8


Horizontal (radial) Error Samples vs. Predicted 90% Ellipsoid Errors
Horizontal (radial) Error (m)
Predicted 90% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8


Horizontal (radial) Error Samples vs. Predicted 50% Ellipsoid Errors
Horizontal (radial) Error (m)
Predicted 50% Ellipsoid Error (m)
0
1
2
3
4
5
6
7
8

Appendix D: Deriving Normalized Error Tolerances

This appendix is related to Section 5.4.2 of the main body of TGD 2c. It provides an overview of the methodology used to derive normalized error tolerances for the three-probability levels: 99%, 90% and 50%. These tolerances are used in both the specification and validation of predicted accuracy requirements. This appendix also presents corresponding non-optimized MATLAB pseudo-code. (Possible optimization includes the use of common functions for the various programs in this and the other appendices.) The general approach makes use of Monte-Carlo simulation – in particular, Random Vector Genration as detailed in TGD2e.

The remainder of this introduction and Sections D.1 and D.2 address the derivation of the normalized error tolerances. The underlying predicted error covariance matrices are assumed scalar multiples (sigma deviation) of the true but unknown error covariance matrix. Section D.3 discusses application of these tolerance to the more general case when the predicted error covariance matrix is not simply a scalar multiple of the true error covariance matrix.

Derivation of normalized error tolerances

The plots of the confidence of passing all three normalized error tests versus sigma deviation or predicted accuracy fidelity (Figures 5.4.2-1 through 5.4.2-12) and the corresponding tables of normalized error tolerances (Tables 5.4.2-2 through 5.4.2-4) were derived via Monte-Carlo simulation. For each plot, statistics were tabulated over the results of 500 different realizations for each “sigma deviation” value, each realization corresponding to the specified number of i.i.d. error samples.

The normalized error test tolerances were derived using Monte-Carlo simulation based on the second set of pseudo-code presented in this appendix (Section D.2). It was used to adjust initial estimates (via substitution of “hard-coded” values) of the normalized error tolerances that were derived using another Monte-Carlo simulation based on the first set of pseudo-code in this appendix (Section D.1). It computed sensitivities of each individual test-level (99, 90, or 50%) to various tolerance values as a function of the “sigma deviation” range. The number of i.i.d. samples was set to 400; thus, essentially are non-factor for these sensitivities.

Subsequent adjustments of these initial estimates were relatively small and resulted in lower tolerance values (easier to pass an individual test) such that all three-level tests then pass. Results were also a function of the assumed and specified number of i.i.d. samples per realization.

Increased probability of success for an individual test is necessary for all three-level tests to pass at the same (or better) probability level. Also, for a given realization, the results of the individual tests are correlated, thus the need for “trial and error” adjustments. For example, if for a given realization (set of error samples), the 99%-level test passes, it is more likely than usual that the 90% level test passes as well, and less likely than usual that the 50% level test (opposite sign test) passes.

The following sensitivity plots correspond to the results of this initial simulation and correspond to the success of an individual normalized horizontal error test assuming 100 samples. Statistics were

tabulated over the results of 500 different realizations for each “sigma deviation” value, each realization corresponding to 100 i.i.d. error samples.

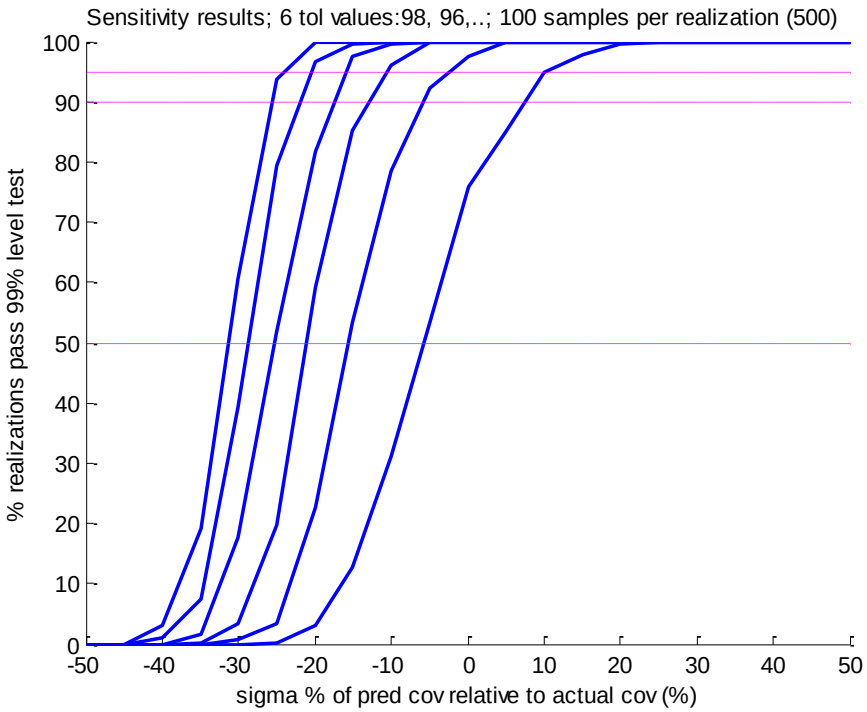


Figure D-1: Sensitivity results for normalized horizontal error test tolerance values; 99%-level test; curve furthest right corresponds to tolerance equal 98%

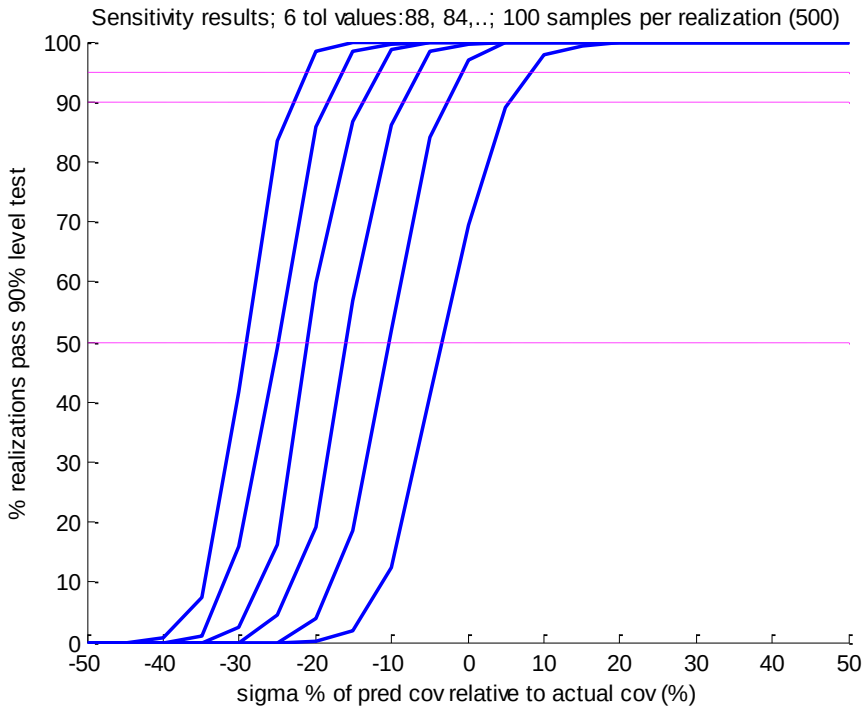


Figure D-2: Sensitivity results for normalized horizontal error test tolerance values; 90%-level test (curve furthest right corresponds to tolerance equal 88%)

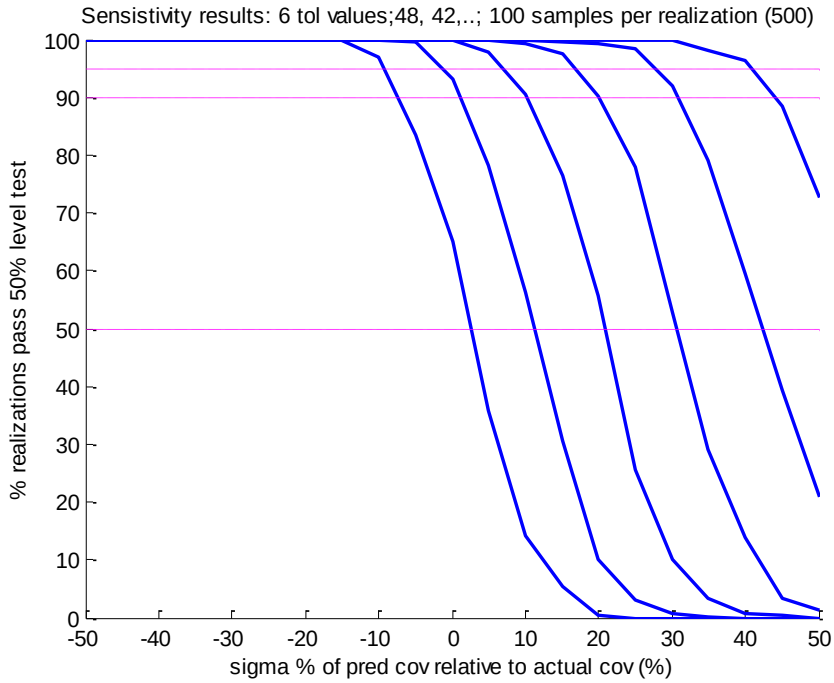


Figure D-3: Sensitivity results for normalized horizontal error test tolerance values; 50%-level test (curve furthest left corresponds to tolerance equal 48%

Note that the horizontal dotted lines in the above plots correspond to confidence at the 99%, 90%, and 50% levels and are there just for convenience; they are not related to the normalized error test tolerance probability-levels of 99%, 90%, and 50%.

Thus, for example, if we are interested in a 90% confidence of individually passing each of the three tests over a sigma deviation range of -15% to +20%, Figures D-1 through D-3, interpolating between curves when necessary, specify approximate tolerances of 94 (99% tolerance probability-level) , 81 (90% tolerance probability-level), and 33 (50% tolerance probability-level), respectively. (For this example, we are only concerned with the -15% sigma deviation end-point for Figures D-1 and D-2, and the +20% sigma deviation end-point for Figure D-3.) These, in turn, become the starting values for adjustment in the second or follow-on Monte-Carlo simulation.

Note that by using the techniques discussed above (with further details “embedded” in the MATLAB pseudo-code), “custom” normalized error tolerance values can also be derived corresponding to different levels of predicted accuracy fidelity other than those assumed in Tables 5.4.2-2 through 5.4.2-4.

D.1 Sensitivity Pseudo-code (individual tests) for Horizontal Errors

```
%"TGD2c_hor_norm_error_sensitivities_ind_tolerances"   5/24/16

% specify desired number of samples, then run multi-level cases:

% compute % of predicted accuracy tests passing validation over a
% specified number of realizations, where each test is based on a
% specified tolerance value, a specified number of samples per realization,
% and a specified sigma deviation;
%
% tol_value cases: (1 to 6: values from 98 down by 2's, from 88 down by 4's,
% and from 48 down by 6's) corresponding to probability levels
% 99, 90, and 50, respectively;

% sigma_level cases (1 to 21: values of predicted error covariance as
% (scaled) % of actual error covariance matrix from -50 to +50%)

n_samples=400
n_real=500
cov_actual=eye(2); % horizontal error covariance matrix,
                   %a generic "actual covariance",
                   % suitable for purpose at hand, since actual cov
                   % "divided out" (to within sigma tolerances) in
                   % computation of normalized errors below; a different
                   % actual cov can be specified if so desired
cov_actual_sqrt=sqrtm(cov_actual);
d_99=3.035;
d_90=2.146;
d_50=1.177;
results=zeros(6,3,21);
```

```
for i2=1:6  %tol values case

   tol_value_99=98-2*(i2-1);
   tol_value_90=88-4*(i2-1);
   tol_value_50=48-6*(i2-1);

   for i4=1:21    % percent sigma case

      sig_factor=-0.5+(i4-1)*0.05;
      cov=(1+sig_factor)^2*cov_actual;

         %begin appropriate validation processing

         per_real_pass_99=zeros(n_real,1);
         per_real_pass_90=zeros(n_real,1);
         per_real_pass_50=zeros(n_real,1);

         for j=1:n_real

            e_norm_99=zeros(n_samples,1);
            e_norm_90=zeros(n_samples,1);
            e_norm_50=zeros(n_samples,1);
            sum_99=0;
            sum_90=0;
            sum_50=0;

            for k=1:n_samples

               s=cov_actual_sqrt*randn(2,1);
               e_norm_90(k,1)=sqrt(s'*cov^-1*s)/d_90;
               e_norm_99(k,1)=(d_90/d_99)* e_norm_90(k,1);
               e_norm_50(k,1)=(d_90/d_50)* e_norm_90(k,1);

               if(e_norm_99(k,1)<1)
                 sum_99=sum_99+1;
               end

               if(e_norm_90(k,1)<1)
                 sum_90=sum_90+1;
               end
              if(e_norm_50(k,1)>1)
                sum_50=sum_50+1;
              end

            end % end samples

            if((100*sum_99/n_samples)>tol_value_99)
               per_real_pass_99(j,1)=1;
            end
            if((100*sum_90/n_samples)>tol_value_90)
               per_real_pass_90(j,1)=1;
            end
```

```
            if((100*sum_50/n_samples)>tol_value_50)
               per_real_pass_50(j,1)=1;
            end

         end % end realizations

         sum_99=0;
         sum_90=0;
         sum_50=0;

         for j=1:n_real

            if(per_real_pass_99(j,1)==1)
               sum_99=sum_99+1;
            end
            if(per_real_pass_90(j,1)==1)
               sum_90=sum_90+1;
            end
            if(per_real_pass_50(j,1)==1)
               sum_50=sum_50+1;
            end

         end

      results(i2,1,i4)=sum_99/n_real;
      results(i2,2,i4)=sum_90/n_real;
      results(i2,3,i4)=sum_50/n_real;

      end  %end i4

   end   %end i2


%plot 99% and 90% and 50% test results: plot
% realization pass test for a the test's specific tolerance
%value as a function of sigma value;

plot1=zeros(21,1);
sig_nmbr=zeros(21,1);
conf_50=zeros(21,1);
conf_90=zeros(21,1);
conf_95=zeros(21,1);

for i=1:21
   sig_nmbr(i,1)=100*(-.5+(i-1)*.05);
   conf_50(i,1)=50;
   conf_90(i,1)=90;
   conf_95(i,1)=95;
end

%99% test results:
figure (1)
```

```
clf  %this clears the figure from the previous run
hold on
for i=1:6
 for j=1:21
      plot1(j,1)=100*results(i,1,j);   %index order: i is tol value
      %case(1-6), probability-level case(1-3), sigma level case (1-21)
 end
plot(sig_nmbr,plot1,'b','LineWidth',2);
end
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Sensitivity results; 6 tol values:98, 96,..;' num2str(n_samples)...
   ' samples per realization (500) '])
xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass 99% level test');
hold off

%90% test results:
figure (2)
clf  %this clears the figure from the previous run
hold on
for i=1:6
 for j=1:21
      plot1(j,1)=100*results(i,2,j);
 end
plot(sig_nmbr,plot1,'b','LineWidth',2);
end
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Sensitivity results; 6 tol values:88, 84,..;' num2str(n_samples)...
   ' samples per realization (500)'])
xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass 90% level test');
hold off

%50% test results:

figure (3)
clf  %this clears the figure from the previous run
hold on
for i=1:6
 for j=1:21
      plot1(j,1)=100*results(i,3,j);
 end
plot(sig_nmbr,plot1,'b','LineWidth',2);
end
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Sensitivity results: 6 tol values;48, 42,..;' num2str(n_samples)...
```

```
   ' samples per realization (500) ']) 
xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass 50% level test');
hold off
```

D.2 Normalized Error Tolerance Pseudo-code (all 3 tests combined) for Horizontal Errors

The following pseudo-code was used to evaluate the effects of normalized error tolerance values on all three (combined) normalized error tests for horziontal errors. The predicted error covariance matrix is assumed a scalar multiple of the true (and known to the simulation) error covariance matrix. The latter is assumed to be the generic 2x2 identity matrix (variances equal 1 meters-squared), but can be set to any valid covariance matrix, if so desired. A few lines of the pseudo-code (currently commented out) did just that: generated a true covariance matrix with variances equal to 1 and 9 meters-squared. Results were virtually identical to using the generic 2x2 identity matrix for the true covariance matrix, as desired and as illustrated by the following plots:

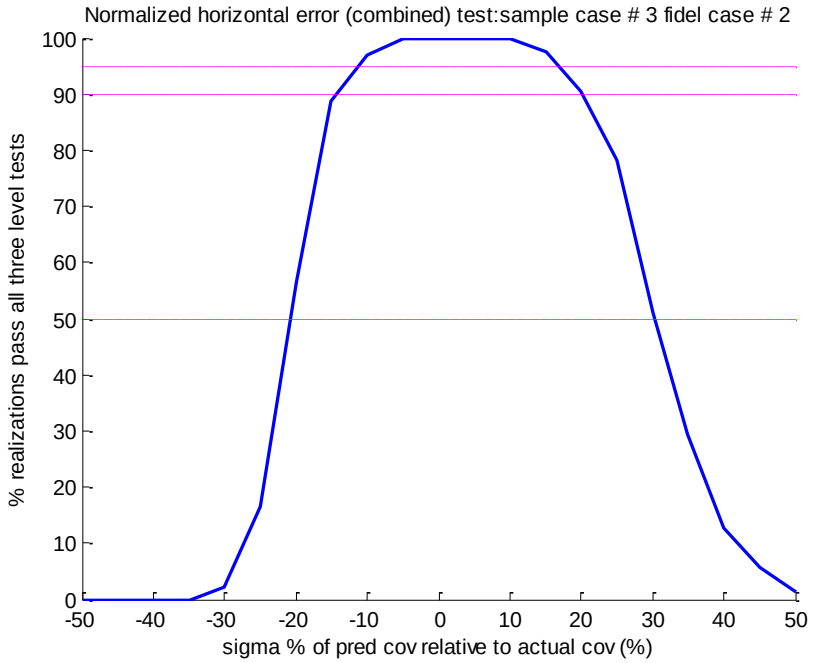


Figure D.2-1: Probability of passing all three normalized error tests; medium predicted accuracy fidelity, 100 i.i.d. samples – true error covariance matrix variances equal 1 meters-squared.

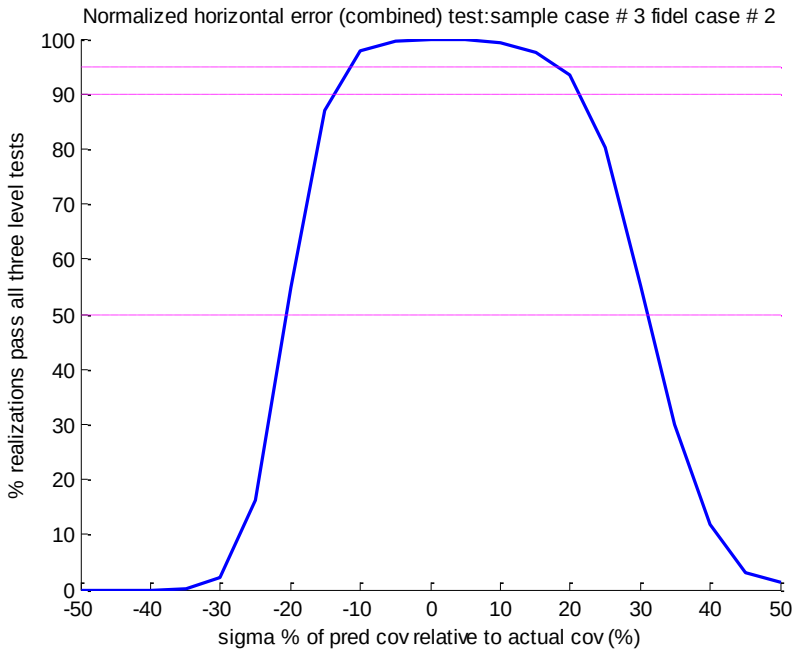


Figure D.2-2: Probability of passing all three normalized error tests; medium predicted accuracy fidelity, 100 i.i.d. samples – true error covariance matrix variances equal 1 and 9 meters-squared.

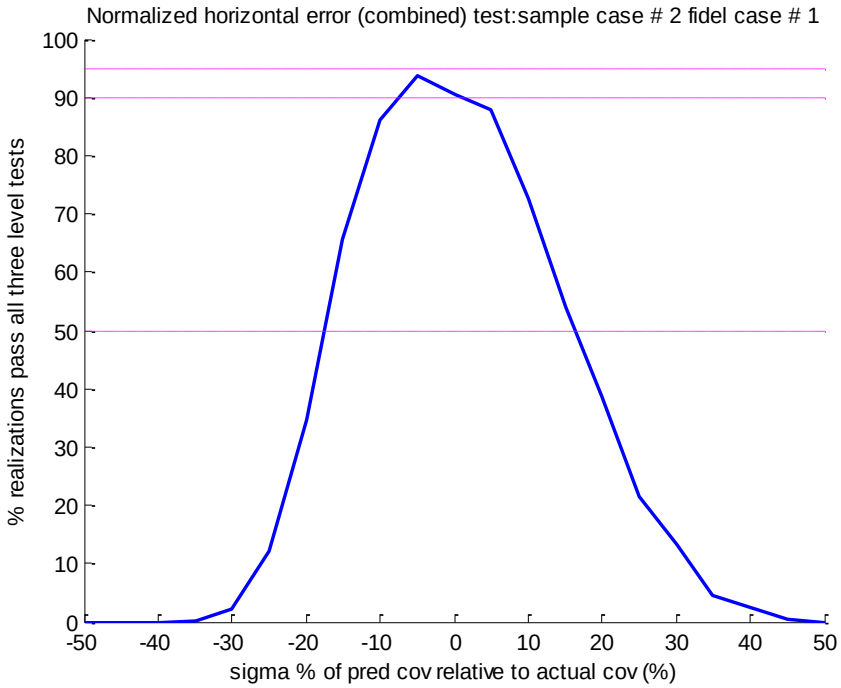


Figure D.2-3: Probability of passing all three normalized error tests; high predicted accuracy fidelity, 50 i.i.d. samples – true error covariance matrix variances equal 1 meters-squared.

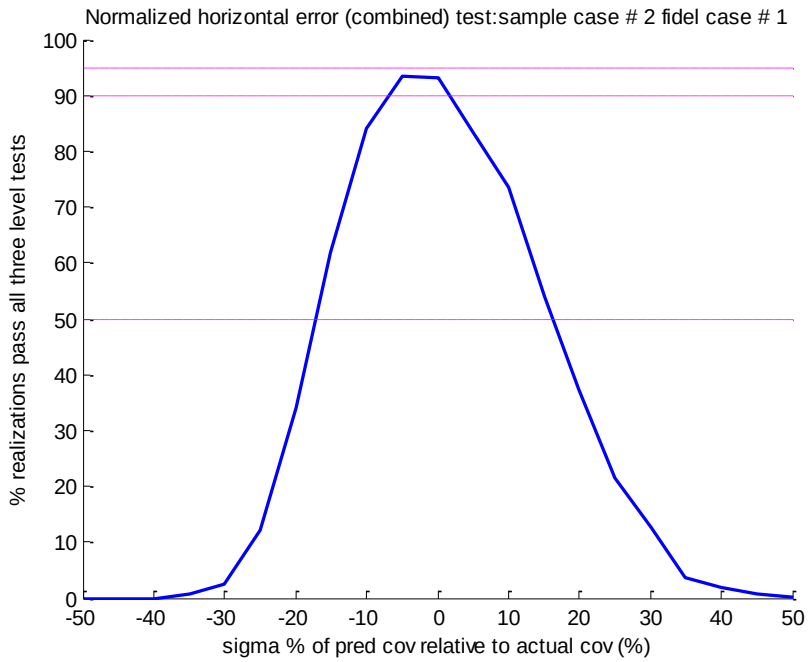


Figure D.2-4: Probability of passing all three normalized error tests; high predicted accuracy fidelity, 50 i.i.d. samples – true error covariance matrix variances equal 1 and 9 meters-squared.

Pseudo-code

```
% "TGD2c_hor_norm_error_combined_tests"   5/25/16

%Confidence that all three probability level test pass for normalized
%errors, parameterized by specified tolerance values, parameterized by
%number of i.i.d. samples, as a function of sigma deviation level.
%Confidence computed as % that all three tests pass taken over numerous
%(nominally 500 realizations).  Note that these are "approximate" results.

%Program used to determine/verify tolerance values that yield at least 90%
%confidence within desired sigma deviation interval, and drop-off as fast
%as possible outside of this interval.

%This program is for horizontal (radial) errors

%Note that toerances below are the defaults, and can also be interpolated
%for either different number of samples or diferent fidelity (sigma range.
%Tolerance values (3 per probability_level) are organized by number of
%samples and predicted accuracy fidelity level.

%Specify desired number of samples and desired predicted accuracy fidelity
%level then run sigma_level cases (21).

%If subsequent results are unsatisfactory, adjust the “hard coded” tolerance values
%in “tol value”and try again until a satisfactory set of tolerances is obtained

%"hard code" the desired sample case # and desired fidelity level#:
```

```
sample_case_nmbr=3  %values 1-4 correspond to 25,50,100,400 samples,
            %respectively

fidel_level_nmbr=1  %values 1-3 correspond to high, medium and low
            %fidelity, respectively

nmbr_samples_choices=[25 50  100 400]
n_samples=nmbr_samples_choices(sample_case_nmbr)

fidelity_choices={'high' 'medium' 'low'}
fidelity=fidelity_choices{fidel_level_nmbr}

%a row in tol_value contains: sample_case_nmbr, fidel_level_nmbr,
%tol_value_99, tol_value_90, tol_value_50

% for a given number of samples and fidelity (row in tol_value), the
% corresponding tolerances are default values and cam be adjusted via this
% program to see results

tol_value=[1   1   90  76  34
       1   2   84  68  24
       1   3   76  54  14
       2   1   93  78  38
       2   2   88  72  30
       2   3   82  60  18
       3   1   95  83  39
       3   2   90  76  30
       3   3   84  64  24
       4   1   97  85  44
       4   2   95  78  36
       4   3   85  65  25];

loc=find(tol_value(:,1)==sample_case_nmbr & tol_value(:,2)==fidel_level_nmbr);
tol_value_99=tol_value(loc,3)
tol_value_90=tol_value(loc,4)
tol_value_50=tol_value(loc,5)

n_real=500  %"hard coded to 500, can be changed if desired

cov_actual=eye(2);   % horizontal error covariance matrix;
            %a generic "actual covariance",
            % suitable for purpose at hand, since actual cov
            % "divided out" (to within sigma tolerances) in
            % computation of normalized errors below; a different
            % actual cov can be specified if so desired

%cov_actual(1,1)=9;  %use non-identity for actual covariance to show results
            %essentially independent of this covariance value, as
            %desired
```

```
cov_actual_sqrt=sqrtm(cov_actual);

d_99=3.035;
d_90=2.146;
d_50=1.177;

results=zeros(21,1);


    for i=1:21    % sigma (%) case

      sig_factor=-0.5+(i-1)*0.05;
      cov=(1+sig_factor)^2*cov_actual;

        %begin appropriate validation processing

        sum_all=0;

        for j=1:n_real

          sum_99=0;
          sum_90=0;
          sum_50=0;

          for k=1:n_samples

            s=cov_actual_sqrt*randn(2,1);%horizontal errors
            e_norm_90=sqrt(s'*cov^-1*s)/d_90;%normalized errors
            e_norm_99=(d_90/d_99)* e_norm_90;
            e_norm_50=(d_90/d_50)* e_norm_90;

           if((e_norm_99<1))  %check if  99% level normalized
                        % error test passes
             sum_99=sum_99+1;
           end
           if((e_norm_90<1))
             sum_90=sum_90+1;
           end
           if((e_norm_50>1))
             sum_50=sum_50+1;
           end

          end   % end samples

           %check if all three tests pass for this realization:
```

```
            if(((100*sum_99/n_samples)>tol_value_99)&&...
                ((100*sum_90/n_samples)>tol_value_90)&&...
                ((100*sum_50/n_samples)>tol_value_50))
              sum_all=sum_all+1;
            end

          end % end realizations

        results(i,1)=sum_all/n_real;   % percent of realizations
                              %where all three tests pass
      end  %end sigma case

%plot test results: plot % realizations pass all three tests for
%the tests' specific tolerance values as a function of sigma value;

plot1=zeros(21,1);
sig_nmbr=zeros(21,1);
conf_50=zeros(21,1); %for info only on plot (50% confidence hor line)
conf_90=zeros(21,1);
conf_95=zeros(21,1);

for i=1:21
   sig_nmbr(i,1)=100*(-.5+(i-1)*.05);
   conf_50(i,1)=50;
   conf_90(i,1)=90;
   conf_95(i,1)=95;
end

figure (1)
clf  %this clears the figure from the previous run
hold on

 for j=1:21
      plot1(j,1)=100*results(j,1);
 end
plot(sig_nmbr,plot1,'b','LineWidth',2);

plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);

axis([-50 50 0 100]);
title(['Normalized horizontal error (combined) test:'...
 num2str(sample_case_nmbr) ' sample case # '...
 num2str(fidel_level_nmbr) ' fidel level #'])
title(['Normalized horizontal error (combined) test:sample case # '...
   num2str(sample_case_nmbr) ' fidel case # ' num2str(fidel_level_nmbr)])

xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass all three level tests');
hold off
```

D.3 Extension of Tolerances to Arbitrary Predicted Error Covariance Matrices

Sections D.1 and D.2 discussed normalized error tolerances assuming that the predicted error covariance matrix was a scalar multiple of the true error covariance matrix – a convenient way to represent predicted accuracy fidelity using "sigma deviation", as discussed in Section 5.4.2. However, predicted error covariance matrices are more general than this when associated with levels of predicted accuracy fidelity, as discussed in Section 5.4.2 as well, and corresponding to Equation (5.4.2-1) and Figure 5.4.2-1 in particular.

For example, assuming medium predicted accuracy fidelity, any predicted error covariance matrix that satisfies the following corresponds to this level of fidelity:

$$(1 + sig\_dev_l)^2 C_X\_true \leq C_X\_pred \leq (1 + sig\_dev_r)^2 C_X\_true, \quad \text{(D.3-1)}$$

where $sig\_dev_l = -0.15$ and $sig\_dev_r = 0.20$.

Also, as illustrated earlier in Figure 5.4.2-1, such a predicted error covariance matrix need not be a scalar multiple of the true error covariance matrix, i.e., their respective probability ellipses need not be the same shape and/or orientation.

Furthermore, assuming Equation (D.3-1) is satisfied and the availability of 100 i.i.d. samples, validation should be successful with the probability of success greater than or equal to the minimum probability across the x-axis range of -15 to +20% in Figure D.2-1 which does assume that the predicted error covariance matrix is a scalar multiple (sigma deviation) of the true error covariance matrix for each value of x. The minimum probability across the x-axis range of -15 to +20% is applicable since the predicted radial computed using the predicted error covariance matrix is bounded by the true radials corresponding to the true error covariance matrix scaled by the -15% and +20% sigma deviation values, also illustrated earlier in Figure 5.4.2-1.

On the other hand, validation should not be successful for any predicted error covariance matrix that satisfies:

$$C_{X_{pred}} < (1 + sig\_dev_{l*})^2 C_X\_true \quad \text{or} \quad (1 + sig\_dev_{r*})^2 C_{X_{true}} < C_X\_pred, \quad \text{(D.3-2)}$$

where $sig\_dev_{l*} < sig\_dev_l$ and $sig\_div_{r*} > sig\_dev_r$, respectively.

The probability of (correct) failure is greater than one minus the maximum probability of success in the corresponding x-axis open intervals $x < sig\_div_{l*}$ and $x > sig\_div_{r*}$, respectively, in Figure D.2-1.

Also, a predicted error covariance matrix can exist that does not satisfy either Equations (D.3-1) or (D.3-2) – is considered in the "roll-off range" between validation success and validation failure.

All of the above situations are covered in the following seven examples based on a modification of the pseudo-code of Section D-2, such that the predicted error covariance matrix is more general that a strict multiple of the true error covariance matrix. For each case, the predicted error covariance matrix

(probability) ellipse (red) is plotted with the $(1 + sig\_dev_l)^2 C_X\_true$ ellipse (light blue) and the $(1 + sig\_dev_r)^2 C_X\_true$ ellipse (blue). All ellipses were computed as 90% probability ellipses.

The corresponding confidence in passing validation (based on 500 independent realizations, each with 100 i.i.d. error samples) is provided in the figure titles. As can be seen, all are consistent with the discussion and content of both the above and Section 5.4.2:

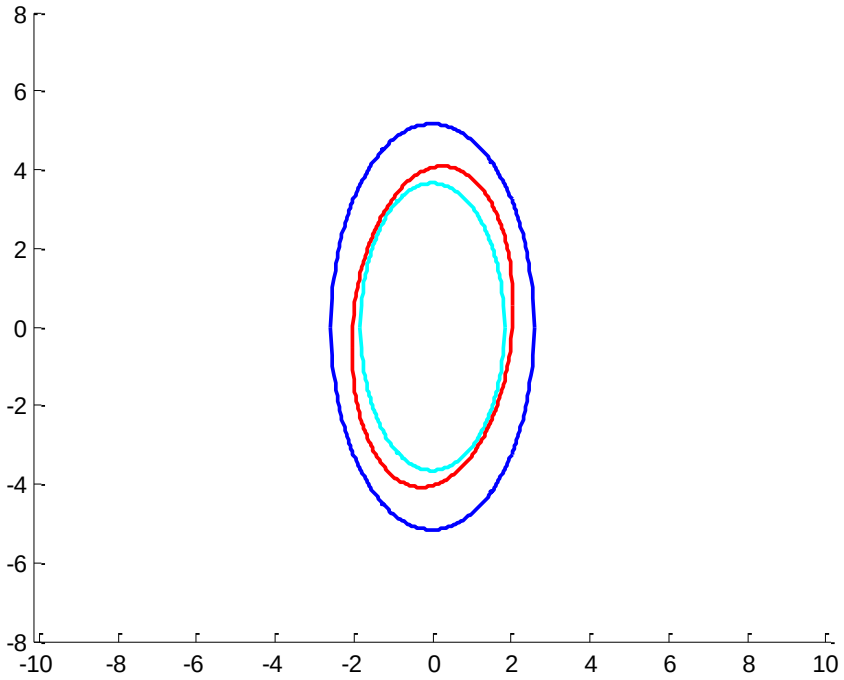


Figure D.3-1: 99.8% probable that validation passes – as desired, i.e., validation should pass; red ellipse bounded by light blue and blue ellipses, i.e., Equation (D.3-1) satisfied

The corresponding error covariance matrices for the above example are generated (simulated) in the pseudo-code at the end of this section (option = 1). Specifically, the actual (true) error covariance matrix is equal to: $cov\_actual = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$ and was used to generate all independent samples of horizontal error. It was also subsequently scaled by sigma deviations equal to -0.15 and +0.20 to yield the corresponding light blue and blue ellipses in the above figure, respectively.

The predicted error covariance matrix is equal to: $cov = \begin{bmatrix} 0.9099 & 0.2358 \\ 0.2358 & 3.6026 \end{bmatrix}$ and was used in the computation of the predicted radials at the three different probability levels (99, 90, and 50%). It corresponds to the red ellipse in the above figure. The predicted error covariance was generated by scaling the actual error covariance matrix by a sigma deviation (aka sig_factor) of -0.05 and subsequently rotating it by theta = -0.03 degrees.

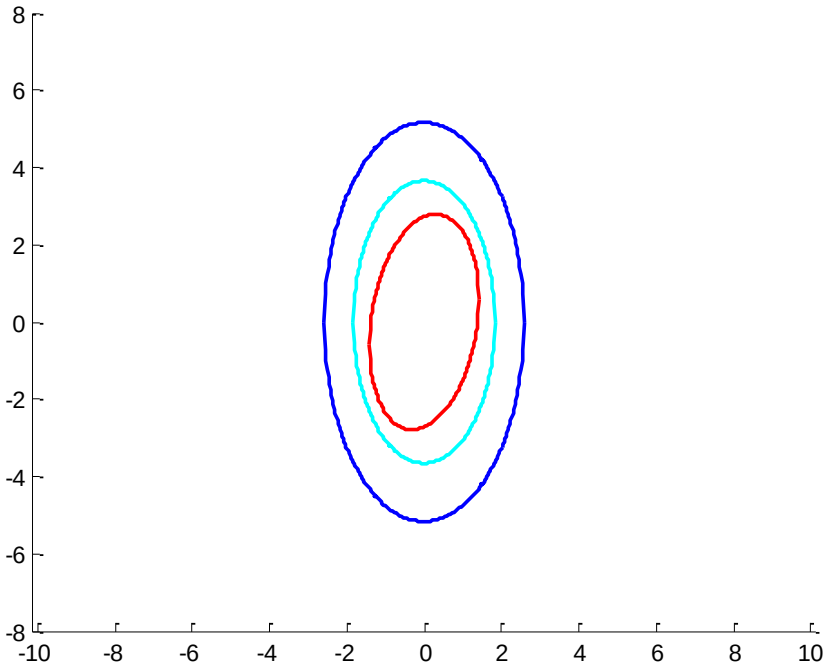


Figure D.3-2: 0 % probable that validation passes – as desired, i.e., validation should fail; red ellipse not bounded (below) by light blue ellipse, i.e., left side of Equation (D.3-2) satisfied

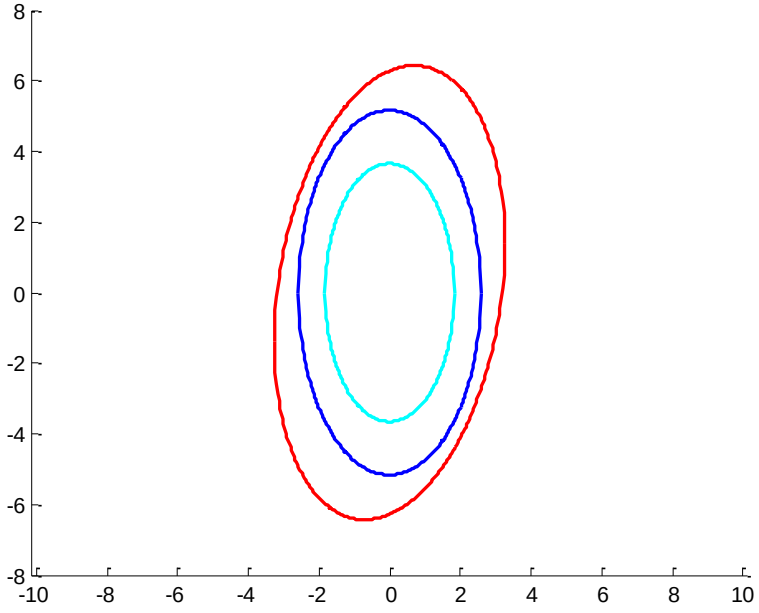


Figure D.3-3: 2.0 % probable that validation passes – as desired, i.e., validation should fail: red ellipse not bounded (above) by blue ellipse, i.e., right side of Equation (D.3-2) satisfied

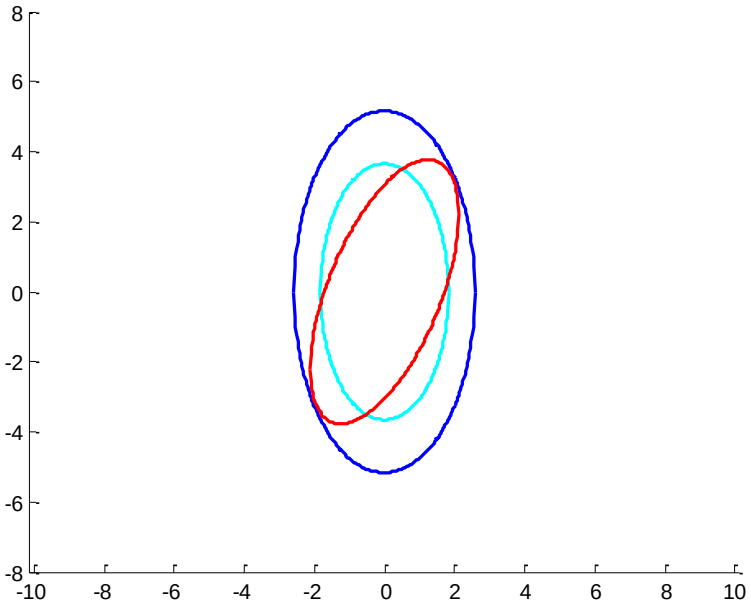


Figure D.3-4: 28.6 % probable that validation passes (roll-off range); neither Equation (D.3-1) or Equation (D.3-2) satisfied

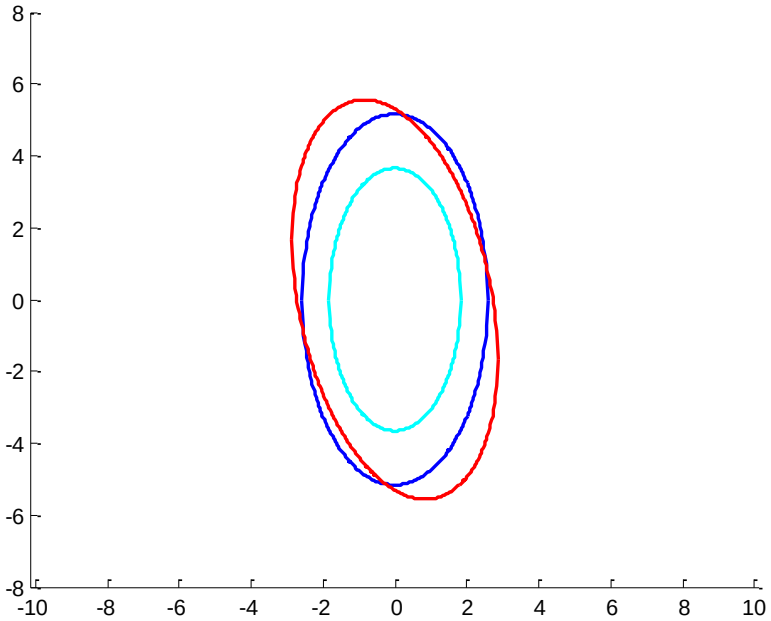


Figure D.3-5: 69.8 % probable that validation passes (roll-off range); neither Equation (D.3-1) or Equation (D.3-2) satisfied

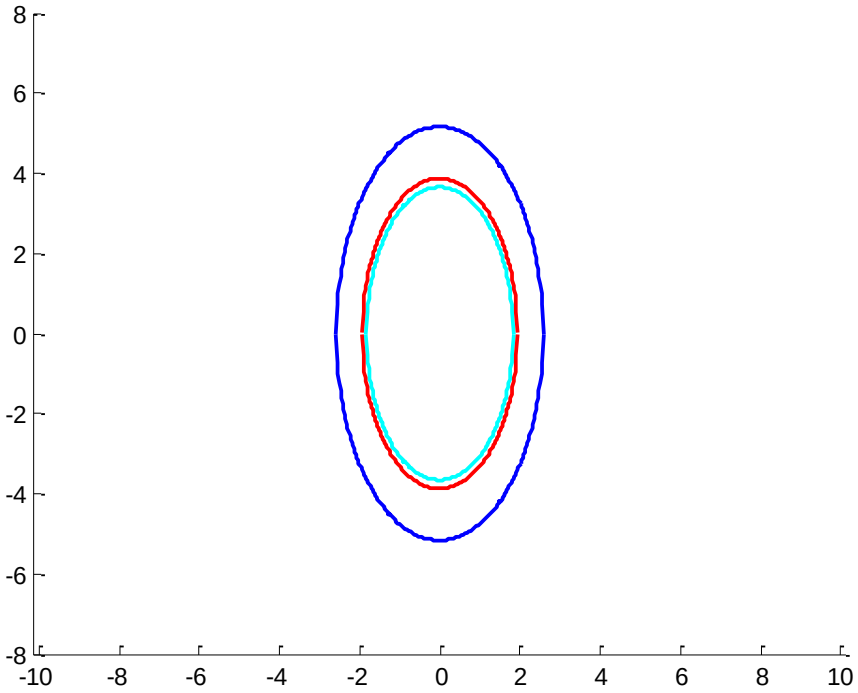


Figure D.3-6: 98.4 % probable that validation passes– as desired, i.e., validation should pass; red ellipse bounded by light blue and blue ellipses, i.e., Equation (D.3-1) satisfied; in this particular case, the predicted error covariance matrix is a scalar multiple of the true or actual error covariance matrix – the corresponding ellipse has the same shape and orientation as the scaled true ellipses and of course, the true ellipse (not shown)

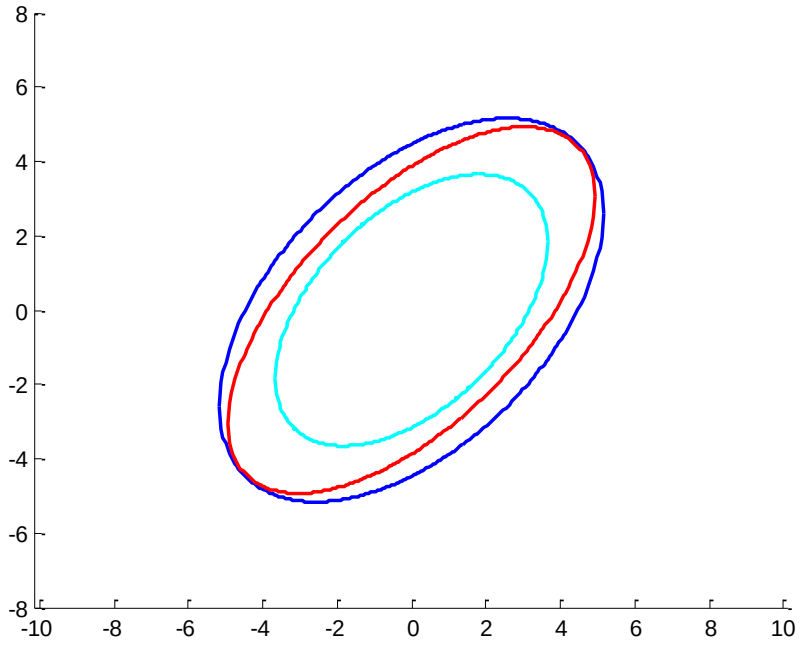


Figure D.3-7: 99.8 % probable that validation passes– as desired, i.e., validation should pass; red ellipse bounded by light blue and blue ellipses, i.e., Equation (D.3-1) satisfied

In the above example (pseudo-code, option = 7), the actual (true) error covariance matrix is equal to: $cov\_actual = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$. The predicted error covariance matrix is equal to: $cov = \begin{bmatrix} 5.2602 & 3.2605 \\ 3.2605 & 5.2603 \end{bmatrix}$.

The non-optimized MATLAB pseudo-code used to compute the confidences corresponding to the above figures via Monte Carlo simulation as well as the plots themselves is provided below (the true or actual error covariance matrix and the predicted error covariance matrix are specified as well):

Pseudo-code

```
% "TGD2c_hor_norm_error_combined_tests_ellipses_plots"   12/27/16

%Assume medium pred accuracy fidelity; plot scaled true ellipses annd pred
%ellipses corresponding to their covariance matrices; compute probability of
%validation success

sample_case_nmbr=3  %values 1-4 correspond to 25,50,100,400 samples,
                    %respectively

fidel_level_nmbr=2  %values 1-3 correspond to high, medium and low
                    %fidelity, respectively

nmbr_samples_choices=[25 50  100 400]
n_samples=nmbr_samples_choices(sample_case_nmbr)
```

```
fidelity_choices={'high' 'medium' 'low'}
fidelity=fidelity_choices{fidel_level_nmbr}

%a row in tol_value contains: sample_case_nmbr, fidel_level_nmbr,
%tol_value_99, tol_value_90, tol_value_50

% for a given number of samples and fidelity (row in tol_value), the
% corresponding tolerances are default values and cam be adjusted via this
% program to see results

tol_value=[1    1   90  76  34
           1    2   84  68  24
           1    3   76  54  14
           2    1   93  78  38
           2    2   88  72  30
           2    3   82  60  18
           3    1   95  83  39
           3    2   90  76  30
           3    3   84  64  24
           4    1   97  85  44
           4    2   95  78  36
           4    3   85  65  25];

loc=find(tol_value(:,1)==sample_case_nmbr                                    &
tol_value(:,2)==fidel_level_nmbr);
tol_value_99=tol_value(loc,3)
tol_value_90=tol_value(loc,4)
tol_value_50=tol_value(loc,5)

n_real=500  %"hard coded to 500, can be changed if desired

cov_actual=eye(2);
cov_actual(2,2)=4;

option=7

if(option==1)    %totally within both bounds
sig_factor=-0.05
theta=-03
end
if(option==2)    %totally inside minimum bound
sig_factor=-0.35
theta=-05
end
if(option==3)    %totally outside maximum bound
sig_factor=0.50
theta=-05
end
if(option==4)     %both in and out minimum bound
sig_factor=-0.10
theta=-15
end
if(option==5)     %both in and out maximum bound
sig_factor=0.30
```

```
theta=07
end
if(option==6)      %totally within both bounds
sig_factor=0-0.10
theta=0
end
if(option==7)      %totally within both bounds; also vastly different
                   %actual covariance than other cases
    cov_actual(1,1)=4
    cov_actual(2,2)=4;
    cov_actual(1,2)=0.5*4;
    cov_actual(2,1)=cov_actual(1,2);
    sig_factor=0.10
    theta=-5
end

cov_actual
cov_actual_sqrt=sqrtm(cov_actual)

rot=zeros(2,2);
rot(1,1)=cosd(theta);
rot(2,2)=rot(1,1);
rot(1,2)=-sind(theta);
rot(2,1)=rot(1,2);

cov=(1+sig_factor)^2*cov_actual;
cov=rot*cov*rot';

cov % predicted covariance matrix

d_99=3.035;
d_90=2.146;
d_50=1.177;
                %begin appropriate validation processing

                sum_all=0;

                for j=1:n_real

                    sum_99=0;
                    sum_90=0;
                    sum_50=0;

                    for k=1:n_samples

                        s=cov_actual_sqrt*randn(2,1);%horizontal errors

                        e_norm_90=sqrt(s'*cov^-1*s)/d_90;%normalized errors
                        e_norm_99=(d_90/d_99)* e_norm_90;
                        e_norm_50=(d_90/d_50)* e_norm_90;

                       if((e_norm_99<1))  %check if  99% level normalized
                                          % error test passes
                           sum_99=sum_99+1;
```

```
                    end
                    if((e_norm_90<1))
                        sum_90=sum_90+1;
                    end
                    if((e_norm_50>1))
                        sum_50=sum_50+1;
                    end

                end   % end samples

                 %check if all three tests pass for this realization:
                if(((100*sum_99/n_samples)>tol_value_99)&&...
                       ((100*sum_90/n_samples)>tol_value_90)&&...
                       ((100*sum_50/n_samples)>tol_value_50))
                    sum_all=sum_all+1;
                end

            end % end realizations

        results=100*sum_all/n_real    % percent of realizations
                                        %where all three tests pass

%Plot all three ellipses
axis_val_x=8;
axis_val_y=8;

A=(1+0.20)^2*cov_actual; %***************
nmbr_x=100;
nmbr_pts=2*nmbr_x+1;
x1=zeros(nmbr_pts,1);
y1=zeros(nmbr_pts,1);
y2=zeros(nmbr_pts,1);
AI=A^-1;
a=AI(1,1);
b=AI(1,2);
c=AI(2,2);
d=-2.15^2;
xlimit=sqrt(-d/(a-b^2/c));
dx=xlimit/nmbr_x;
for i=1:nmbr_pts
    x1(i,1)=-xlimit+(i-1)*dx;
    y1(i,1)=-2*(b/c)*x1(i,1);
    y1(i,1)=y1(i,1)-sqrt(4*(b/c)^2*x1(i,1)^2-4*(a/c)*x1(i,1)^2-4*d/c);
    y1(i,1)=y1(i,1)/2;
    y2(i,1)=-2*(b/c)*x1(i,1);
    y2(i,1)=y2(i,1)+sqrt(4*(b/c)^2*x1(i,1)^2-4*(a/c)*x1(i,1)^2-4*d/c);
    y2(i,1)=y2(i,1)/2;
end
figure(1)
clf
hold on
plot(x1,y1,'b',x1,y2,'b','LineWidth',2)
axis([-axis_val_x axis_val_x -axis_val_y axis_val_y])

A=(1-0.15)^2*cov_actual;    %**********************
```

```
nmbr_x=100;
nmbr_pts=2*nmbr_x+1;
x1=zeros(nmbr_pts,1);
y1=zeros(nmbr_pts,1);
y2=zeros(nmbr_pts,1);
AI=A^-1;
a=AI(1,1);
b=AI(1,2);
c=AI(2,2);
d=-2.15^2;
xlimit=sqrt(-d/(a-b^2/c));
dx=xlimit/nmbr_x;
for i=1:nmbr_pts
    x1(i,1)=-xlimit+(i-1)*dx;
    y1(i,1)=-2*(b/c)*x1(i,1);
    y1(i,1)=y1(i,1)-sqrt(4*(b/c)^2*x1(i,1)^2-4*(a/c)*x1(i,1)^2-4*d/c);
    y1(i,1)=y1(i,1)/2;
    y2(i,1)=-2*(b/c)*x1(i,1);
    y2(i,1)=y2(i,1)+sqrt(4*(b/c)^2*x1(i,1)^2-4*(a/c)*x1(i,1)^2-4*d/c);
    y2(i,1)=y2(i,1)/2;
end
plot(x1,y1,'cy',x1,y2,'cy','LineWidth',2);

A=cov;    %*********************
nmbr_x=100;
nmbr_pts=2*nmbr_x+1;
x1=zeros(nmbr_pts,1);
y1=zeros(nmbr_pts,1);
y2=zeros(nmbr_pts,1);
AI=A^-1;
a=AI(1,1);
b=AI(1,2);
c=AI(2,2);
d=-2.15^2;
xlimit=sqrt(-d/(a-b^2/c));
dx=xlimit/nmbr_x;
for i=1:nmbr_pts
    x1(i,1)=-xlimit+(i-1)*dx;
    y1(i,1)=-2*(b/c)*x1(i,1);
    y1(i,1)=y1(i,1)-sqrt(4*(b/c)^2*x1(i,1)^2-4*(a/c)*x1(i,1)^2-4*d/c);
    y1(i,1)=y1(i,1)/2;

    y2(i,1)=-2*(b/c)*x1(i,1);
    y2(i,1)=y2(i,1)+sqrt(4*(b/c)^2*x1(i,1)^2-4*(a/c)*x1(i,1)^2-4*d/c);
    y2(i,1)=y2(i,1)/2;
end
plot(x1,y1,'r',x1,y2,'r','LineWidth',2);
axis equal
hold off
```

Appendix E: Deriving Normalized Error Tolerances for Use with Scalar Accuracy Metrics Psuedo-code

This appendix supports Section 5.4.3 of the main body of TGD 2c. The (non-optimized) MATLAB pseudo-code contained in this appendix was used to derive Figures 5.4.3-1 through 5.4.3-6, the confidence of passing all three of the normalized error tests when scalar accuracy metrics are used to normalize errors.

Normalized error tolerance pseudo-code (all 3 tests combined) for horizontal error using scalar accuracy metrics

```
% "TGD2c_hor_norm_error_via_scalar_acc_metrics_combined_tests"    5/25/16

%Uses CE_90 and scaled versions of same for CE_99 and CE_50 assuming
%specified actual and specified assumed value for ratio_sqrt_eigen.

%Like "TGD2c_hor_norm_error_combined_tests" except scalar accuracy metrics
%are used for normalization.
%Note that these are "approximate" results.

%This program is for horizontal (radial) errors

%Note that toerances below are the defaults, and can be interpolated for
%either different number of samples or diferent fidelity (sigma range).

%Specify desired number of samples and desired predicted accuracy fidelity
% level then run sigma_level cases (21).

%"hard code" the desired sample case # and desired fidelity level#:

rse_level=6    %true sqrt eigenvalues ratio equals 1-(level-1)*.1, where
          %level can equal 1 through 10

ratio_sqrt_eigen=1-(rse_level-1)*.1

rse_level_est_range=3 %estimated max rse_level for scaling when only
           %CE_90 is computed and without knowledge of rse_level)
ratio_sqrt_eigen_est_range=1-(rse_level_est_range-1)*.1

sample_case_nmbr=3  %values 1-4 correspond to 25, 50, 100, and 400 samples,
              %respectively

fidel_level_nmbr=1  %values 1-3 correspond to high, medium and low fidelity,
              %respectively

nmbr_samples_choices=[25 50  100 400]
n_samples=nmbr_samples_choices(sample_case_nmbr)

fidelity_choices={'high' 'medium' 'low'}
fidelity=fidelity_choices{fidel_level_nmbr}
```

```
%a row in tol_value contains:
%sample_case_nmbr, fidel_level_nmbr,tol_value_99, tol_value_90,tol_value_50

% for a given number of samples and fidelity (row in tol_value),
% the corresponding tolerances are default values

tol_value=[1   1   90  76  34
       1   2   84  68  24
       1   3   76  54  14
       2   1   93  78  38
       2   2   88  72  30
       2   3   82  60  18
       3   1   95  83  39
       3   2   90  76  30
       3   3   84  64  24
       4   1   97  85  44
       4   2   95  78  36
       4   3   85  65  25];

loc=find(tol_value(:,1)==sample_case_nmbr & tol_value(:,2)==fidel_level_nmbr);
tol_value_99=tol_value(loc,3)
tol_value_90=tol_value(loc,4)
tol_value_50=tol_value(loc,5)

n_real=500  %"hard coded to 500, can be changed if desired

cov_actual=eye(2);   % horizontal radials; a generic "actual covariance",
                     % suitable for purpose at hand, since actual cov
                     % "divided out" (to within sigma toleracnes) in
                     % computation of normalized errors below; a different
                     % actual cov can be specified if so desired

cov_actual(2,2)=ratio_sqrt_eigen^2;

cov_actual_sqrt=sqrtm(cov_actual);

P=eye(3);

%specified d levels as a function of rse_level:

d_99=[3.035 2.90 2.79 2.72 2.67 2.63 2.61 2.59 2.58 2.58]
d_90=[2.146 2.04 1.95 1.86 1.79 1.74 1.70 1.67 1.66 1.65]
d_50=[1.177 1.12 1.06 1.00 0.93 0.87 0.81 0.75 0.71 0.68]

results=zeros(21,1);
results_CE=zeros(21,1);
results_CE_s=zeros(21,1);



    for i=1:21    % sigma (%) case
```

```
sig_factor=-0.5+(i-1)*0.05;
cov=(1+sig_factor)^2*cov_actual;
P(1,1)=cov(1,1);
P(2,2)=cov(2,2);

   %begin appropriate validation processing

   sum_all=0;
   sum_all_CE=0;
   sum_all_CE_s=0;

   for j=1:n_real

      sum_99=0;
      sum_90=0;
      sum_50=0
      sum_99_CE=0;
      sum_90_CE=0;
      sum_50_CE=0
      sum_99_CE_s=0;
      sum_90_CE_s=0;
      sum_50_CE_s=0;

      for k=1:n_samples

         s=cov_actual_sqrt*randn(2,1);   %horizontal errors

         e_norm_90=sqrt(s'*cov^-1*s)/d_90(1);%norm errors
         e_norm_99=(d_90(1)/d_99(1))* e_norm_90;
         e_norm_50=(d_90(1)/d_50(1))* e_norm_90;

        %check if  99% level normalized error test passes,
        %etc, using predicted radials computed above:
        if((e_norm_99<1))
           sum_99=sum_99+1;
        end
        if((e_norm_90<1))
           sum_90=sum_90+1;
        end
        if((e_norm_50>1))
           sum_50=sum_50+1;
        end

         eh=sqrt(s(1,1)^2+s(2,1)^2);
         [CE_90 LE_90]=CE_50_90_99_compute(P,90);

         e_norm_90=eh/CE_90;    %normalized errors
         e_norm_99=(d_90(rse_level_est_range)/...
            d_99(rse_level_est_range))* e_norm_90;
         e_norm_50=(d_90(rse_level_est_range)/...
            d_50(rse_level_est_range))* e_norm_90;
```

```
  %check if  99% level normalized error test passes,
  %etc, using scalar acc metrics computed above
  %assuming an est rse level:
  if((e_norm_99<1))
     sum_99_CE_s=sum_99_CE_s+1;
  end
  if((e_norm_90<1))
     sum_90_CE_s=sum_90_CE_s+1;
  end
  if((e_norm_50>1))
     sum_50_CE_s=sum_50_CE_s+1;
  end

   e_norm_90=eh/CE_90;    %normalized errors
   e_norm_99=(d_90(rse_level)/...
      d_99(rse_level))* e_norm_90;
   e_norm_50=(d_90(rse_level)/...
      d_50(rse_level))* e_norm_90;

  %check if  99% level normalized error test passes,
  %etc, using scalar acc metrics computed above
  %the actual rse level:
  if((e_norm_99<1))
     sum_99_CE=sum_99_CE+1;
  end
  if((e_norm_90<1))
     sum_90_CE=sum_90_CE+1;
  end
  if((e_norm_50>1))
     sum_50_CE=sum_50_CE+1;
  end


end   % end samples

%check if all three tests pass for this realization:

if(((100*sum_99/n_samples)>tol_value_99)&&...
     ((100*sum_90/n_samples)>tol_value_90)&&...
     ((100*sum_50/n_samples)>tol_value_50))
   sum_all=sum_all+1;
end
```

```
            if(((100*sum_99_CE_s/n_samples)>tol_value_99)&&...
                ((100*sum_90_CE_s/n_samples)>tol_value_90)&&...
                ((100*sum_50_CE_s/n_samples)>tol_value_50))
              sum_all_CE_s=sum_all_CE_s+1;
            end

             if(((100*sum_99_CE/n_samples)>tol_value_99)&&...
                ((100*sum_90_CE/n_samples)>tol_value_90)&&...
                ((100*sum_50_CE/n_samples)>tol_value_50))
              sum_all_CE=sum_all_CE+1;
            end

          end % end realizations

          results(i,1)=sum_all/n_real;  % percent of realizations
                                %where all three tests pass
          results_CE_s(i,1)=sum_all_CE_s/n_real; % percent of
             %realizations where all three tests pass using scaled CE
          results_CE(i,1)=sum_all_CE/n_real; % percent of realizations
               %where all three tests pass using properly computed CE

      end  %end sigma case

%plot 99% and 90% and 50% test results:
%plot % realization pass test for a the test's specific tolerance
%value as a function of sigma value;

plot1=zeros(21,1);
plot2=zeros(21,1);
plot3=zeros(21,1);
sig_nmbr=zeros(21,1)
conf_50=zeros(21,1); %for info only on plot (50% confidence hor line)
conf_90=zeros(21,1);
conf_95=zeros(21,1);

for i=1:21
   sig_nmbr(i,1)=100*(-.5+(i-1)*.05);
   conf_50(i,1)=50;
   conf_90(i,1)=90;
   conf_95(i,1)=95;
end

for j=1:21
      plot1(j,1)=100*results(j,1);
      plot2(j,1)=100*results_CE_s(j,1);
      plot3(j,1)=100*results_CE(j,1);
 end
```

```
figure (1)
clf  %this clears the figure from the previous run
hold o
plot(sig_nmbr,plot1,'b','LineWidth',2);
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Normalized horizontal test - radials (b): sample case # '...
   num2str(sample_case_nmbr) ' fidel case # ' num2str(fidel_level_nmbr)]
xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass all three level tests');
hold off



figure (2)
clf  %this clears the figure from the previous run
hold on
plot(sig_nmbr,plot2,'r','LineWidth',2);
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Normalized horizontal test - CE s (b): sample case # '...
   num2str(sample_case_nmbr) ' fidel case # ' num2str(fidel_level_nmbr)])
xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass all three level tests');
hold off



figure (3)
clf  %this clears the figure from the previous run
hold on
plot(sig_nmbr,plot3,'g','LineWidth',2);
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Normalized horizontal test - CE (b): sample case # '...
   num2str(sample_case_nmbr) ' fidel case # ' num2str(fidel_level_nmbr)])
xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass all three level tests');
hold off
```

```
figure (4)
clf  %this clears the figure from the previous run
hold on

plot(sig_nmbr,plot1,'b',sig_nmbr,plot2,'r',sig_nmbr,plot3,'g',...
   'LineWidth',2);
plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
   '--m','LineWidth',1);
axis([-50 50 0 100]);
title(['Norm horizontal test - radials(b),CEs(r),CE(g): sample case # '...
   num2str(sample_case_nmbr) ' fidel case # ' num2str(fidel_level_nmbr)])

xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass all three level tests');
hold off
```

Appendix F: Accuracy and Predicted Accuracy Validation Examples Pseudo-code

This Appendix contains the (non-optimized) MATLAB pseudo-code used to generate the examples of Sections 5.2.3 and 5.2.4 of the main body of TGD 2c.

See the following pseudo-code for the actual predicted error covariance matrices simulated (e.g. P1 and P2) and corresponding actual (true) error covariance matrices (e.g. P_actual), the latter used to generate the independent samples of error.

Accuracy validation examples and predicted accuracy validation examples pseudo-code:

```
%Simulation program for accuracy and predicted accuracy validation examples
%"TGD2c_commerical_imagery_spec_and_validation_example"; 5/24/16

%This code computes both vertical and horizontal (radial) errors normalized
%by corresponding predicted scalar accuracy metrics (LEXX, CEXX) and
%horizontal and 3d radial errors normalized by predicted radials for
%predicted accuracy validation.

%It also computes the 90% least-upper-bound for vertical and horizontal
%(radial) error 90% percentiles for accuracy validation.

%the input parameters controlling the simulation are hard-coded for %simplicity

n_samples=100  %Specify desired number of samples

% set assumed actual-to-predicted error covariance matrix scale factor:

sf=0.95^2 %somewhat optimistic predicted accuracy
%sf=0.8^2  %optimistic predicted accuracy
%sf=1.03^2  %slightly pessimistic predicted accuracy"


%Specify 2 different "basic" actual (true) 3x3 error covariance matrices for
%"variability" and modelling within "operational constraints"; values
%essentially arbitrary but must correspond to valid covariance matrices (sym
%and pos definite) for 3d geolocation errors:

P1=zeros(3,3);
P1(1,1)=1.2;
P1(2,2)=1.1;
P1(3,3)=1.3;
P1(1,2)=.2*sqrt(P1(1,1)*P1(2,2));
P1(2,1)=P1(1,2);
P1(1,3)=.1*sqrt(P1(1,1)*P1(3,3));
P1(3,1)=P1(1,3);
P1(2,3)=.8*sqrt(P1(2,2)*P1(3,3));
P1(3,2)=P1(2,3);
P1=3*P1;
```

```
P2=zeros(3,3);
P2(1,1)=2.2;
P2(2,2)=1.6;
P2(3,3)=1.8;
P2(1,2)=.2*sqrt(P2(1,1)*P2(2,2));
P2(2,1)=P2(1,2);
P2(1,3)=.1*sqrt(P2(1,1)*P2(3,3));
P2(3,1)=P2(1,3);
P2(2,3)=.8*sqrt(P2(2,2)*P2(3,3));
P2(3,2)=P2(2,3);
P2=3*P2;


P1  %output their values, their corresponding CE90's, and their eigenvalues
P2
[CE_90_P1 LE_90_P1]=CE_50_90_99_compute(P1,90)
[CE_90_P2 LE_90_P2]=CE_50_90_99_compute(P2,90)
eig_P1=eig(P1)
eig_P2=eig(P2)
%compute sqrt of smallest to largest eigenvalue in upper 2x2 covariance
%matrix:
PH1=zeros(2,2);
PH2=zeros(2,2);
for ii=1:2
   for jj=1:2
      PH1(ii,jj)=P1(ii,jj);
      PH2(ii,jj)=P2(ii,jj);
   end
end
eig_PH1=eig(PH1)
ratio1=sqrt(eig_PH1(1)/eig_PH1(2))
eig_PH2=eig(PH2)
ratio2=sqrt(eig_PH2(1)/eig_PH2(2))
```

```
%initialize statistical compilations and related variables

dz_samples=zeros(n_samples,1);
LE_50_samples=zeros(n_samples,1);
LE_90_samples=zeros(n_samples,1);
LE_99_samples=zeros(n_samples,1);

dH_samples=zeros(n_samples,1);
CE_50_samples=zeros(n_samples,1);
CE_90_samples=zeros(n_samples,1);
CE_99_samples=zeros(n_samples,1);

dX_samples=zeros(n_samples,1);
P_50_r_samples=zeros(n_samples,1);
P_90_r_samples=zeros(n_samples,1);
P_99_r_samples=zeros(n_samples,1);

PH_50_r_samples=zeros(n_samples,1);
PH_90_r_samples=zeros(n_samples,1);
PH_99_r_samples=zeros(n_samples,1);

LE_50_percent=0;
LE_90_percent=0;
LE_99_percent=0;
CE_50_percent=0;
CE_90_percent=0;
CE_99_percent=0;
P_50_r_percent=0;
P_90_r_percent=0;
P_99_r_percent=0;
PH_50_r_percent=0;
PH_90_r_percent=0;
PH_99_r_percent=0;

XH=zeros(2,1);
PH=zeros(2,2);
xaxis=zeros(n_samples,1);
y45=zeros(n_samples,1);


for i=1:n_samples  %loop over number of samples  (one "realization")

%pick one of the two basic actual covariances and then mod slightly for
%more realism:

k=mod(i,2);
P_actual=P2;
if(k==0)
   P_actual=P1;
end
```

```
%del=0;   %make all the covariances the same
%del=.05; %mod as a scalar multiple in %
del=.01;
for k=1:3
   d=del*P_actual(k,k)*randn(1,1);
   P_actual(k,k)=P_actual(k,k)+d;
end
temp=eig(P_actual);
for j=1:3
   if(temp(j,1)<=0)
      bad_eigen=temp(j,1)  %check for modified covariance invalid and %printout as warning
   end
end

%generates random error samples consistent with actual error covariance
%matrix:
X=sqrtm(P_actual)*randn(3,1);   %random 3D errors


P=sf*P_actual;  %compute (Mig-type) predicted error covariance matrix for %current error samples

%set actual 2d horizontal errors appropriately
for j=1:2
   XH(j,1)=X(j,1);
end
%set predicted horizontal error covariance matrix appropriately
for ii=1:2
   for jj=1:2
      PH(ii,jj)=P(ii,jj);
   end
end

%compute actual radial errors;
dX=sqrt(X(1,1)^2+X(2,1)^2+X(3,1)^2);
dH=sqrt(X(1,1)^2+X(2,1)^2);
dz=sqrt(X(3,1)^2);

%compute predicted CE50,CE90,CE99 and LE50,LE90,LE99 scalar accuracy metrics:
%(function assumes full 3x3 %covariance matrix)
[CE_50 LE_50]=CE_50_90_99_compute(P,50);
[CE_90 LE_90]=CE_50_90_99_compute(P,90);
[CE_99 LE_99]=CE_50_90_99_compute(P,99);
```

```
%compile vertical normalized error results based on scalar acuracy metrics
%for current sample
%(note:results same as if used vertical predicted radials)
dz_samples(i,1)=dz;
LE_50_samples(i,1)=LE_50;
LE_90_samples(i,1)=LE_90;
LE_99_samples(i,1)=LE_99;
if(dz<=LE_50)
   LE_50_percent=LE_50_percent+1;
end
if(dz<=LE_90)
   LE_90_percent=LE_90_percent+1;
end
if(dz<=LE_99)
   LE_99_percent=LE_99_percent+1;
end

%compile horizontal normalized error results based on scalar accuracy metrics
%for current sample
dH_samples(i,1)=dH;
CE_50_samples(i,1)=CE_50;
CE_90_samples(i,1)=CE_90;
CE_99_samples(i,1)=CE_99;
if(dH<=CE_50)
   CE_50_percent=CE_50_percent+1;
end
if(dH<=CE_90)
   CE_90_percent=CE_90_percent+1;
end
if(dH<=CE_99)
   CE_99_percent=CE_99_percent+1;
end

%compile (3d) radial normalized error results based on predicted radials for
%current sample
P_50_r=dX*1.538/sqrt((X'*(P^-1)*X));   %compute predicted radials
P_90_r=dX*2.500/sqrt((X'*(P^-1)*X));
P_99_r=dX*3.368/sqrt((X'*(P^-1)*X));
dX_samples(i,1)=dX;
P_50_r_samples(i,1)=P_50_r;
P_90_r_samples(i,1)=P_90_r;
P_99_r_samples(i,1)=P_99_r;
if(dX<=P_50_r)
   P_50_r_percent=P_50_r_percent+1;
end
if(dX<=P_90_r)
   P_90_r_percent=P_90_r_percent+1;
end
if(dX<=P_99_r)
   P_99_r_percent=P_99_r_percent+1;
end
```

```
%compile horizontal normalized error results based on predicted radials for
%current sample
PH_50_r=dH*1.177/sqrt((XH'*(PH^-1)*XH));   %compute predicted radials
PH_90_r=dH*2.146/sqrt((XH'*(PH^-1)*XH));
PH_99_r=dH*3.035/sqrt((XH'*(PH^-1)*XH));
PH_50_r_samples(i,1)=PH_50_r;
PH_90_r_samples(i,1)=PH_90_r;
PH_99_r_samples(i,1)=PH_99_r;
if(dH<=PH_50_r)
   PH_50_r_percent=PH_50_r_percent+1;
end
if(dH<=PH_90_r)
   PH_90_r_percent=PH_90_r_percent+1;
end
if(dH<=PH_99_r)
   PH_99_r_percent=PH_99_r_percent+1;
end

max=13;  %13 meters max for x-axis and y_axis
xaxis(i,1)=(i-1)*max/n_samples; %set up various "boilerplate" plotting info
y45(i,1)=xaxis(i,1);

end   %end samples loop

%output predicted accuracy results for all three probability-levels
%for both normalization based on scalar accuracy metrics (first 6 outputs)
%and normalization based on predicted radials (last 6 outputs); note that
%passing the correspondsing predicted accuracy validation test requires that
%the results are greatere than or equal to the specified normalized error
%tolerance requirements (not included in code)

LE_50_percent=1-LE_50_percent/n_samples  %subtract from 1 since test is
%"above line", not below line
LE_90_percent=LE_90_percent/n_samples
LE_99_percent=LE_99_percent/n_samples
CE_50_percent=1-CE_50_percent/n_samples
CE_90_percent=CE_90_percent/n_samples
CE_99_percent=CE_99_percent/n_samples
P_50_r_percent=1-P_50_r_percent/n_samples
P_90_r_percent=P_90_r_percent/n_samples
P_99_r_percent=P_99_r_percent/n_samples
PH_50_r_percent=1-PH_50_r_percent/n_samples
PH_90_r_percent=PH_90_r_percent/n_samples
PH_99_r_percent=PH_99_r_percent/n_samples
```

```
%(Note: results for validation of accuracy based on order statistics are
%generated after code to output Plot #14 below)

%Plot results in terms of un-normalized error samples vs. appropriate
%predicted accuracy samples (predicted scalar accuracy metric or radial at
%appropriate probability level)

figure (1)
clf  %this clears the figure from the previous run
hold on
scatter(LE_50_samples,dz_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('z radial error samples vs. predicted 50% LE')
xlabel('predicted LE 50 (m)')
ylabel('radial z error (m)');
hold off

figure (2)
clf
hold on
scatter(LE_90_samples,dz_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('z radial error samples vs. predicted 90% LE')
xlabel('predicted LE 90 (m)')
ylabel('radial z error (m)');
hold off

figure (3)
clf
hold on
scatter(LE_99_samples,dz_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('z radial error samples vs. predicted 99% LE')
xlabel('predicted LE 99 (m)')
ylabel('z radial error (m)');
hold off


figure (4)
clf
hold on
scatter(CE_50_samples,dH_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('horizontal radial error samples vs. predicted 50% CE')
xlabel('predicted CE 50 (m)')
ylabel('horizontal radial error (m)');
hold off
```

```
figure (5)
clf
hold on
scatter(CE_90_samples,dH_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('horizontal radial error samples vs. predicted 90% CE')
xlabel('predicted CE 90 (m)')
ylabel('horizontal radial error (m)');
hold off

figure (6)
clf
hold on
scatter(CE_99_samples,dH_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('horizontal radial error samples vs. predicted 99% CE 99')
xlabel('predicted CE 99 (m)')
ylabel('horizontal radial error (m)');
hold off

figure (7)
clf
hold on
scatter(P_50_r_samples,dX_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('radial (3d) error samples vs. predicted 50% ellipsoid radial')
xlabel('predicted 50% ellipsoid radial (m)')
ylabel('radial (3d) error (m)');
hold off

figure (8)
clf
hold on
scatter(P_90_r_samples,dX_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('radial (3d) error samples vs. predicted 90% ellipsoid radial')
xlabel('predicted 90% ellipsoid radial (m)')
ylabel('radial (3d) error (m)');
hold off
```

```
figure (9)
clf
hold on
scatter(P_99_r_samples,dX_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('radial (3d) error samples vs. predicted 99% ellipsod radial')
xlabel('predicted 99% ellipsoid radial (m)')
ylabel('radial (3d) error (m)');
hold off

figure (10)
clf
hold on
scatter(PH_50_r_samples,dH_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('horizontal radial error samples vs. predicted 50% ellipse radial')
xlabel('predicted 50% ellipse radial (m)')
ylabel('horizontal radial error (m)');
hold off

figure (11)
clf
hold on
scatter(PH_90_r_samples,dH_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('horizontal radial error samples vs. predicted 90% ellipse radial')
xlabel('predicted 90% ellipse radial (m)')
ylabel('horizontal radial error (m)');
hold off

figure (12)
clf
hold on
scatter(PH_99_r_samples,dH_samples)
plot(xaxis,y45);
axis([0 max 0 max]);
title('horizontal radial error samples vs. predicted 99% ellipse radial')
xlabel('predicted 99% ellipse radial (m)')
ylabel('horizontal radial error (m)');
hold off
```

```
%add the "three line" plots for predicted radials only;

y_small_slope=zeros(n_samples,1);
y_large_slope=zeros(n_samples,1);

for i=1:n_samples
y_small_slope(i,1)=(1.538/2.500)*xaxis(i,1);
y_large_slope(i,1)=(3.368/2.500)*xaxis(i,1);
end

figure (13)
clf
hold on
scatter(P_90_r_samples,dX_samples)
plot(xaxis,y45,'b');
plot(xaxis,y_small_slope,'r');
plot(xaxis,y_large_slope,'m');
axis([0 max 0 max]);
title('radial (3d) errors samples vs. predicted 90% ellipsoid radial: 50%,... 90%, 99% lines (r, b, m)')
xlabel('predicted 90% ellipsoid radial (m)')
ylabel('radial (3d) error (m)');
hold off

y_small_slope=zeros(n_samples,1);
y_large_slope=zeros(n_samples,1);
cH_above_line1=0;
cH_below_line2=0;
cH_below_line3=0;

for i=1:n_samples
y_small_slope(i,1)=(1.177/2.146)*xaxis(i,1);
y_large_slope(i,1)=(3.035/2.146)*xaxis(i,1);
end

figure (14)
clf
hold on
scatter(PH_90_r_samples,dH_samples)
plot(xaxis,y45,'b');
plot(xaxis,y_small_slope,'r');
plot(xaxis,y_large_slope,'m');
axis([0 max 0 max]);
title('90% 2D Error Ellipse samples: 50% line (r), 90% line (b), 99% line... (m)')
xlabel('predicted 90% Ellipse radial (aka 90% predicted horizontal radial) (m)')
ylabel('horizontal radial error (m)');
hold off
```

```
%ORDER STATISTIC RESULTS FOR VALDIDATION OF ACCURACY

%assume n_samples=100;
%order sample 95 for lub of 90th percentile; order sample 90 for best
%estimate of 90th percentile
%(see TGD2b for order sample indices assuming a different number of samples)

sorted_dz=sort(dz_samples);
sorted_dH=sort(dH_samples);
sorted_dX=sort(dX_samples);

lub_dz=sorted_dz(95,1)
lub_dH=sorted_dH(95,1)
lub_dX=sorted_dX(95,1)

best_est_dz=sorted_dz(90,1)
best_est_dH=sorted_dH(90,1)
best_est_dX=sorted_dX(90,1)

%note that passing the corresponding accuracy validation test requires that
%the a lub computed above is less than the corresponding specified
%requirement (not included in code)

lub_dz_p=zeros(n_samples,1);
lub_dH_p=zeros(n_samples,1);
lub_dX_p=zeros(n_samples,1);
be_dz_p=zeros(n_samples,1);
be_dH_p=zeros(n_samples,1);
be_dX_p=zeros(n_samples,1);


samp=zeros(n_samples,1);
for i=1:n_samples
   samp(i,1)=i;
   lub_dz_p(i,1)=lub_dz;
   lub_dH_p(i,1)=lub_dH;
   lub_dX_p(i,1)=lub_dX;
   be_dz_p(i,1)=best_est_dz;
   be_dH_p(i,1)=best_est_dH;
   be_dX_p(i,1)=best_est_dX;
end
```

```
figure (15)
clf
hold on
scatter(samp,dz_samples)
plot(samp,lub_dz_p,'m',samp,be_dz_p,'--r');
axis([1 100 0 max]);
title('vertical (radial) error samples; lub (m), best est (r)');
xlabel('sample number')
ylabel('error (m)');
hold off

figure (16)
clf
hold on
scatter(samp,dH_samples)
plot(samp,lub_dH_p,'m',samp,be_dH_p,'--r')
axis([1 100 0 max]);
title('horizontal (radial) error sample; lub (m), best est (r)')
xlabel('sample number')
ylabel('error (m)');
hold off

figure (17)
clf
hold on
scatter(samp,dX_samples);
plot(samp,lub_dX_p,'m',samp,be_dX_p,'--r')
axis([1 100 0 max]);
title('radial (3d) error samples; lub (m), best est (r)')
xlabel('sample number')
ylabel('error (m)');
hold off
```

Appendix G: Processing Correlated Error Samples Pseudo-code

Section G.1 of this appendix contains pseudo-code that supports Section 5.6 and its various sub-sections. Section G.2 presents details of further options regarding the algorithm/equation for sub-method 1 of the representative error samples approach, including applications to vertical and 3d errors.

G.1 Pseudo-code

There are 4 sets of (non-optimized) pseudo-code. The first set of pseudo-code (Section G.1.1) was used to generate the histogram results of Section 5.6.1 regarding the importance of using i.i.d. error samples in general. It basically generates uncorrelated (i.i.d.) error samples as well as correlated error samples and then implements the baseline accuracy validation code (Sections 4.1/5.1) which uses order statistics and assumes the use of i.i.d. error samples.

The second set of pseudo-code (Section G.1.2) was used to generate the confidence plots for passing all three normalized error test, using the "all error samples approach" (n_cor_sample correlated errors per i.i.d. sample error) of Section 5.6.3 as the pseudo-code's external option 1, or the "representative error samples" approach of Section 5.6.4 as the pseudo-code's external option 2, with the pseudo-code's suboptions 1 and 2 corresponding to sub-methods 1 and 2, respectively, of the "representative error samples" approach. Predicted covariance is assumed a function of sigma deviation.

The third set of pseudo-code (Section G.1.3) was used to generate the specific example corresponding to sub-method 1 of the "representative error samples" approach that was presented in Section 5.6.4.1.

The fourth set of code (Section G.1.4) was used to generate comparison results between sub-methods (or sub-options) 1 and 2 of the "representative error samples" approach. It addresses values of the normalized error test and the 90% probability level only for simplicity, which is associated with predicted accuracy (validation). It also presents validation of accuracy results as well. This pseudo-code supports the results of Section 5.6.4.3.

The various sets of pseudo-code use Monte-Carlo techniques and simulate underlying geolocation errors consistent with a multi-variate Gaussian probability distribution. Although any corresponding algorithms that were based on order statistics and corresponding to accuracy (as opposed to predicted accuracy) require no such assumption, Monte-Carlo simulation does require an assumed probability distribution and multi-variate Gaussian was selected as most appropriate.

G.1.1 Importanace of i.i.d. samples in general

Pseudo-code in support of the first part of Section 5.6.1, followed by pseudo-code for the second part of that section, i.e., last example of the (correlated sub-collection) case:

```
%"TGD2c_cor_sample_effects"          9/29/16

%horizontal errors; 90th percentile; 90% confidence for lub


n_samples=60

n_cor_cases=3

cor_cases=zeros(n_cor_cases,1);

cor_cases(1,1)=0;
cor_cases(2,1)=50;
cor_cases(3,1)=90;

cor_cases

cov_true=eye(2);
%eh_90_true=2.141
ratio=0.8;
cov_true(2,2)=1.5625;%ratio=0.8
eh_90_true=1.9472*1.25;%ratio=0.25

cov_true
eh_90_true

order_be=54
order_90=58

n_real=1000


for i=1:n_cor_cases

        cor=cor_cases(i,1)/100
        kk=2*n_samples;
        cov=zeros(kk,kk);

         be=zeros(n_real,1);
         lub_90=zeros(n_real,1);
```

```
 for j=1:n_real

            for k1=1:n_samples
                for k2=1:n_samples
                    cor_temp=1;
                    if(k1~=k2)
                        cor_temp=cor;
                    end
                    cov((k1-1)*2+1,(k2-1)*2+1)=cor_temp*cov_true(1,1);
                    cov((k1-1)*2+1,(k2-1)*2+2)=cor_temp*cov_true(1,2);
                    cov((k1-1)*2+2,(k2-1)*2+1)=cor_temp*cov_true(2,1);
                    cov((k1-1)*2+2,(k2-1)*2+2)=cor_temp*cov_true(2,2);
                end
            end

            samples=sqrtm(cov)*randn(kk,1);
            radial_samples=zeros(n_samples,1);

            for k1=1:n_samples
                radial_samples(k1,1)=sqrt(samples((k1-
1)*2+1,1)^2+samples((k1-1)*2+2,1)^2);
            end

            radial_samples=sort(radial_samples);

            be(j,1)=radial_samples(order_be,1);
            lub_90(j,1)=radial_samples(order_90,1);

        end % end realizations

        %be
        %lub_90

        lub_90_asc=sort(lub_90);
        kk=floor(0.10*n_real);
        lub_90_asc_90=lub_90_asc(kk,1)

        lub_pad=lub_90-eh_90_true*ones(n_real,1);
        lub_pad=sort(lub_pad);
        kk=floor(0.90*n_real);
        lub_pad_90=lub_pad(kk,1);
        lub__pad_90_rel=lub_pad_90/eh_90_true

        x=0.5:0.1:5.5;
        figure((i-1)*2+1)
        %axis([0.5 4.5 0 +inf]);
        hist(be,x);
        figure((i-1)*2+2)
        hist(lub_90,x);


end %cor_cases
```

```
%"TGD2c_cor_sample_effects_2"          9/29/16

%horizontal errors; 90th percentile; 90% confidence for lub

%alternate


n_samples=60
cor=50

cov_true=eye(2);
%eh_90_true=2.141
ratio=0.8;
cov_true(2,2)=1.5625;%ratio=0.8
eh_90_true=1.9472*1.25;%ratio=0.25

cov_true
eh_90_true

order_be=54
order_90=58

n_real=1000


for i=1:1

        cor=cor/100;
        kk=2*n_samples;
        cov=zeros(kk,kk);

         be=zeros(n_real,1);
         lub_90=zeros(n_real,1);

        for j=1:n_real

            for m=1:6
            for k1=1+(m-1)*10:1+m*10-1
                for k2=1+(m-1)*10:1+m*10-1
                    cor_temp=1;
                    if(k1~=k2)
                        cor_temp=cor;
                    end
                    cov((k1-1)*2+1,(k2-1)*2+1)=cor_temp*cov_true(1,1);
                    cov((k1-1)*2+1,(k2-1)*2+2)=cor_temp*cov_true(1,2);
                    cov((k1-1)*2+2,(k2-1)*2+1)=cor_temp*cov_true(2,1);
                    cov((k1-1)*2+2,(k2-1)*2+2)=cor_temp*cov_true(2,2);
                end
            end
            end
```

```
            %cov

            samples=sqrtm(cov)*randn(kk,1);
            radial_samples=zeros(n_samples,1);

            for k1=1:n_samples
                radial_samples(k1,1)=sqrt(samples((k1-
1)*2+1,1)^2+samples((k1-1)*2+2,1)^2);
            end

            radial_samples=sort(radial_samples);

            be(j,1)=radial_samples(order_be,1);
            lub_90(j,1)=radial_samples(order_90,1);

        end % end realizations

        %be
        %lub_90

        lub_90_asc=sort(lub_90);
        kk=floor(0.10*n_real);
        lub_90_asc_90=lub_90_asc(kk,1)

        lub_pad=lub_90-eh_90_true*ones(n_real,1);
        lub_pad=sort(lub_pad);
        kk=floor(0.90*n_real);
        lub_pad_90=lub_pad(kk,1);
        lub__pad_90_rel=lub_pad_90/eh_90_true

        x=0.5:0.1:5.5;
        figure((i-1)*2+1)
        %axis([0.5 4.5 0 +inf]);
        hist(be,x);
        figure((i-1)*2+2)
        hist(lub_90,x);


end %cor_cases
```

G.1.2 Confidence plots for passing normalized error tests

```
% "TGD2c_hor_norm_error_combined_tests_all"   7/19/16

%Confidence that all three probability level test pass for normalized
%errors,parameterized by specified tolerance values, parameterized by
%number of i.i.d. samples, as a function of sigma deviation level.
%Confidence computed as % that all three tests pass taken over numerous
%(nominally 500 realizations).  Note that these are "approximate" results.
```

```
%Program used to determine/verify tolerance values that yield at least 90%
%confidence within desired sigma deviation interval, and drop-off as fast
%as possible outside of this interval.

%This program is for horizontal (radial) errors.
%This version is "non-optimized" in support
%of "fast experimenation".

%In this extended (all) version of the program, number of i.i.d. "regular"
%samples is extended to a total of samples*n_cor_samples, where the group
%of n_cor_samples per "regular" samples are correlated by cor.
%See "TGD2c_hor_norm_error_combined_test" for orginal
%version and more original  comment statements.

%It has two "external" options for using the extra
%samples per regular sample:(1) process all n_samples*n_cor_samples samples
%separately but use non-standard tolerance values, i.e, "All
%samples approach", and (2) consolidate
%n_cor_samples regular samples into one representative sample.  This later
%approach also has two suboptions.

%Finally note that all processing, regardless the method/options
%designated, computenormalized error at the appropriate probability levels
%for the correspondonding <1 or >1 tests.  The normalized error can be
%easily suballocated to the appropriate (non-normalized) radial error and
%corresponding prodicted radial as well, but is not included in this code
%which is primarily concerned with plotting the confidence of passing all
%three normalized error tests vs. sigma deviation


ext_option=1
sub_option=1  %(value corresponds to ext option 2's suboption 1 or 2)

n_cor_samples=6
cor=0.8
%cor=0.5      %note that cor=0.5 selected (hard coded) for ext option 1 and
%100 samples; a form of "customization" of the normalized error tolerances

sample_case_nmbr=3  %values 1-4 correspond to 25,50,100,400 samples,
                    %respectively; can not be larger than 3 if for
                    %ext option=1 as customized tolerances have not
                    %been generated yet for this option

fidel_level_nmbr=1  %values 1-3 correspond to high, medium and low
                    %fidelity, respectively

nmbr_samples_choices=[25 50  100 400]
n_samples=nmbr_samples_choices(sample_case_nmbr)

fidelity_choices={'high' 'medium' 'low'}
fidelity=fidelity_choices{fidel_level_nmbr}
```

```
%a row in tol_value contains: sample_case_nmbr, fidel_level_nmbr,
%tol_value_99, tol_value_90, tol_value_50

tol_value=[1    1   90  76  34
           1    2   84  68  24
           1    3   76  54  14
           2    1   93  78  38
           2    2   88  72  30
           2    3   82  60  18
           3    1   95  83  39
           3    2   90  76  30
           3    3   84  64  24
           4    1   97  85  44
           4    2   95  78  36
           4    3   85  65  25];

loc=find(tol_value(:,1)==sample_case_nmbr & tol_value(:,2)==...
    fidel_level_nmbr);

if(ext_option==1) %use tolerance corresponding to next highest sample case
                  %as a temproary way" to customize tolerances
    loc=find(tol_value(:,1)==(sample_case_nmbr+1) & tol_value(:,2)==...
        fidel_level_nmbr);
end

tol_value_99=tol_value(loc,3)
tol_value_90=tol_value(loc,4)
tol_value_50=tol_value(loc,5)

n_real=500  %"hard coded to 500, can be changed if desired

cov_actual=eye(2);    % horizontal radials; a generic "actual covariance"
                      % common to each horizontal error sample
cov_actual(2,2)=4;    %generalize somewhat to different sigmas
cov_actual_sqrt=sqrtm(cov_actual);

cov_actual_group=eye(2*n_cor_samples); %covariance for all n_cor_sample
                                       %corrrelated horizontal errors

for i=1:n_cor_samples
    for j=1:n_cor_samples
        for ii=1:2
            for jj=1:2
                k=(i-1)*2+ii;
                l=(j-1)*2+jj;
                cov_actual_group(k,l)=cov_actual(ii,jj);
                if(i~=j)
                    cov_actual_group(k,l)=cor*cov_actual_group(k,l);
                end
            end
        end
    end
end
```

```
cov_actual_group_sqrt=sqrtm(cov_actual_group);
%cov_actual_group;
%cov_actual_group_sqrt

d_99=3.035;
d_90=2.146;
d_50=1.177;

results=zeros(21,1);
s_temp=zeros(2,1);
e_norm_50=zeros(n_cor_samples,1);
e_norm_90=zeros(n_cor_samples,1);
e_norm_99=zeros(n_cor_samples,1);

        for i=1:21    % sigma (%) case

            sig_factor=-0.5+(i-1)*0.05;
            cov=(1+sig_factor)^2*cov_actual;

                %begin appropriate validation processing

                sum_all=0;

                for j=1:n_real

                    sum_99=0;
                    sum_90=0;
                    sum_50=0;

                    for k=1:n_samples

                        s=cov_actual_group_sqrt*randn(2*n_cor_samples,1);

                        if(ext_option==1)   %ext option 1

                        for m=1:n_cor_samples

                        s_temp(1,1)=s((m-1)*2+1,1);
                        s_temp(2,1)=s((m-1)*2+2,1);

                        e_norm_50(m,1)=sqrt(s_temp'*cov^-1*s_temp)/1.177;
                        e_norm_90(m,1)=sqrt(s_temp'*cov^-1*s_temp)/2.146;
                        e_norm_99(m,1)=sqrt(s_temp'*cov^-1*s_temp)/3.035;

                            if((e_norm_99(m,1)<1))  %check if  99% level
                                        % normalized error test passes
                                sum_99=sum_99+1;
                            end
                            if((e_norm_90(m,1)<1))
                                sum_90=sum_90+1;
                            end
```

```
    if((e_norm_50(m,1)>1))
        sum_50=sum_50+1;
    end

    tot_samp=n_samples*n_cor_samples;

end

end  %end ext_option=1

if(ext_option==2)   %ext option 2

if(sub_option==1)  %Suboption 1 processing

mean_X_samp=zeros(2,1);
sigma_X_samp=zeros(2,2);
tempX=zeros(2,1);

for m=1:n_cor_samples

s_temp(1,1)=s((m-1)*2+1,1);
s_temp(2,1)=s((m-1)*2+2,1);

mean_X_samp(1,1)=mean_X_samp(1,1)+s_temp(1,1);
mean_X_samp(2,1)=mean_X_samp(2,1)+s_temp(2,1);
end

mean_X_samp=mean_X_samp/n_cor_samples;

for m=1:n_cor_samples
sigma_X_samp=sigma_X_samp+(s_temp-mean_X_samp)*...
    (s_temp-mean_X_samp)';
end

sigma_X_samp=sigma_X_samp/(n_cor_samples-1);

tempX(1,1)=sqrt(mean_X_samp(1,1)^2+...
sigma_X_samp(1,1)^2);
tempX(2,1)=sqrt(mean_X_samp(2,1)^2+...
sigma_X_samp(2,1)^2);


temp=sqrt(tempX'*(cov^-1)*tempX);  %cov assumed
%common and diagonal across samples, if not take avg
%cov and zero off-diagonals

avg_50=temp/1.177;
avg_90=temp/2.146;
avg_99=temp/3.035;

end   %end suboption 1 unique processing
```

```
    if (sub_option==2) %Suboption 2 processing
     mean_X_samp=zeros(2,1);
     sigma_X_samp=zeros(2,2);
     tempX=zeros(2,1);

     for m=1:n_cor_samples

     s_temp(1,1)=s((m-1)*2+1,1);
     s_temp(2,1)=s((m-1)*2+2,1);

     mean_X_samp(1,1)=mean_X_samp(1,1)+s_temp(1,1);
     mean_X_samp(2,1)=mean_X_samp(2,1)+s_temp(2,1);
     end

     mean_X_samp=mean_X_samp/n_cor_samples;

    for m=1:n_cor_samples
    sigma_X_samp=sigma_X_samp+(s_temp-mean_X_samp)*...
        (s_temp-mean_X_samp)';
    end

    sigma_X_samp=sigma_X_samp/(n_cor_samples-1);

    tempX(1,1)=sqrt((abs(mean_X_samp(1,1))+0.5*...
        sqrt(sigma_X_samp(1,1)))^2);
    tempX(2,1)=sqrt((abs(mean_X_samp(2,1))+0.5*...
        sqrt(sigma_X_samp(2,2)))^2);

    temp=sqrt(tempX'*(cov^-1)*tempX);  %cov assumed
    %common and diagonal across samples, if not take avg
    %cov and zero off-diagonals

     avg_50=temp/1.177;
     avg_90=temp/2.146;
     avg_99=temp/3.035;

  end %end Suboption2 unique processing

       if((avg_99<1))  %check if  99% level normalized
                     % error test passes
           sum_99=sum_99+1;
       end
       if((avg_90<1))
           sum_90=sum_90+1;
       end
       if((avg_50>1))
           sum_50=sum_50+1;
       end

       tot_samp=n_samples;

    end  %end ext_option=2

end   % end samples
```

```
                    %check if all three tests pass for this realization:
                    if(((100*sum_99/tot_samp)>tol_value_99)&&...
                           ((100*sum_90/tot_samp)>tol_value_90)&&...
                           ((100*sum_50/tot_samp)>tol_value_50))
                        sum_all=sum_all+1;
                    end

                end % end realizations

            results(i,1)=sum_all/n_real;    % percent of realizations
                                            %where all three tests pass
        end  %end sigma case

%plot test results: plot % realizations pass all three tests for
%the tests' specific tolerance values as a function of sigma value;

plot1=zeros(21,1);
sig_nmbr=zeros(21,1);

conf_50=zeros(21,1); %for info only on plot (50% confidence hor line)
conf_90=zeros(21,1);
conf_95=zeros(21,1);

for i=1:21
    sig_nmbr(i,1)=100*(-.5+(i-1)*.05);
    conf_50(i,1)=50;
    conf_90(i,1)=90;
    conf_95(i,1)=95;
end

figure (1)
clf  %this clears the figure from the previous run
hold on

 for j=1:21
        plot1(j,1)=100*results(j,1);
 end
plot(sig_nmbr,plot1,'b','LineWidth',2);

plot(sig_nmbr,conf_50,'--m',sig_nmbr,conf_90,'--m',sig_nmbr,conf_95,...
    '--m','LineWidth',1);

axis([-50 50 0 100]);
title(['Normalized horizontal error (combined) test:'...
 num2str(sample_case_nmbr) ' sample case # '...
 num2str(fidel_level_nmbr) ' fidel level #'])
title(['Normalized horizontal error (combined) test:sample case # '...
    num2str(sample_case_nmbr) ' fidel case # ' num2str(fidel_level_nmbr)])

xlabel('sigma % of pred cov relative to actual cov (%)')
ylabel('% realizations pass all three level tests');
hold off
```

G.1.3 Specfic example of sub-method 1 of "representative error samples" approach

```
%"TGD2c_cor_sample_effects_3"         9/30/16

%horizontal errors; 90th percentile; 90% confidence for lub via
%representative samples (sub-method 1)

%not efficient code so as to print intermediate results

%also perform predicted accuracy validation using representative samples

n_cor_groups=60
n_cor_samples=6

cor=90

cov_true=eye(2);
%eh_90_true=2.141
ratio=0.8;
cov_true(2,2)=1.5625;%ratio=0.8
eh_90_true=1.9472*1.25;%ratio=0.25

cov_true
eh_90_true

order_be=54   %per n_cor_groups after representative samples computed
order_90=58

%in order to pass valdiation using 60 i.i.d. samples and high pred
%aaccuracy fidelity need approximately Y_h_99=93 ,Y_h_90=78, Y_h_50=38 per
%interpolation (really not needed) of Table 4.2-1:

cov_pred=(1-0.03)^2*cov_true  %assume sigma deviaton -3%, should pass
%cov_pred=(1-0.25)^2*cov_true   % should fail pred acc valid
%cov_pred=(1+0.25)^2*cov_true   % should fail pred acc valid

Y_h_99=0.93
Y_h_90=0.79
Y_h_50=0.38

rep_radial=zeros(n_cor_groups,1);
rep_sample_mean_x=zeros(n_cor_groups,1);
rep_sample_mean_y=zeros(n_cor_groups,1);
rep_sample_sigma_x=zeros(n_cor_groups,1);
rep_sample_sigma_y=zeros(n_cor_groups,1);

rep_horiz=zeros(2,n_cor_groups);

cor=cor/100;
kk=2*n_cor_samples;
```

```
for i=1:n_cor_groups

        %generate samples and sample stats for each cor group

        cov=zeros(kk,kk);

            for k1=1:n_cor_samples
                for k2=1:n_cor_samples
                    cor_temp=1;
                    if(k1~=k2)
                        cor_temp=cor;
                    end
                    cov((k1-1)*2+1,(k2-1)*2+1)=cor_temp*cov_true(1,1);
                    cov((k1-1)*2+1,(k2-1)*2+2)=cor_temp*cov_true(1,2);
                    cov((k1-1)*2+2,(k2-1)*2+1)=cor_temp*cov_true(2,1);
                    cov((k1-1)*2+2,(k2-1)*2+2)=cor_temp*cov_true(2,2);
                end
            end

            samples=sqrtm(cov)*randn(kk,1);

            samples_x=zeros(n_cor_samples,1);
            samples_y=zeros(n_cor_samples,1);

            sum_x=0;
            sum_y=0;
            for k1=1:n_cor_samples
                samples_x(k1,1)=samples((k1-1)*2+1,1);
                samples_y(k1,1)=samples((k1-1)*2+2,1);
                sum_x=sum_x+samples_x(k1,1);
                sum_y=sum_y+samples_y(k1,1);
            end
            sample_mean_x=sum_x/n_cor_samples;
            sample_mean_y=sum_y/n_cor_samples;
            sum_x=0;
            sum_y=0;
            for k1=1:n_cor_samples
                sum_x=sum_x+(samples((k1-1)*2+1,1)-sample_mean_x)^2;
                sum_y=sum_y+(samples((k1-1)*2+2,1)-sample_mean_y)^2;
            end
            sample_sigma_x=sqrt(sum_x/(n_cor_samples-1));
            sample_sigma_y=sqrt(sum_y/(n_cor_samples-1));

           rep_radial_sample=sqrt(sample_mean_x^2+sample_sigma_x^2+...
               sample_mean_y^2+sample_sigma_y^2);

           if(i==1||i==2||i==60)
               i
               samples_x
               samples_y
           end

           rep_sample_mean_x(i,1)=sample_mean_x;
           rep_sample_mean_y(i,1)=sample_mean_y;
           rep_sample_sigma_x(i,1)=sample_sigma_x;
```

```
            rep_sample_sigma_y(i,1)=sample_sigma_y;
            rep_horiz(1,i)=sqrt(sample_mean_x^2+sample_sigma_x^2);
            rep_horiz(2,i)=sqrt(sample_mean_y^2+sample_sigma_y^2);
            rep_radial(i,1)=rep_radial_sample;

end


rep_sample_mean_x
rep_sample_mean_y
rep_sample_sigma_x
rep_sample_sigma_y
rep_radial

%rep_horiz

rep_radial_sort=sort(rep_radial);
%compute accuracy validation results using rep hor radial error samples:
be=rep_radial_sort(54)
lub_90=rep_radial_sort(58)

%now perform predicted accuracy validation using rep hor radial error
%samples:

eX=zeros(2,1);
norm_rad_at_99=zeros(n_cor_groups,1);
norm_rad_at_90=zeros(n_cor_groups,1);
norm_rad_at_50=zeros(n_cor_groups,1);

sum1=0;
sum2=0;
sum3=0;

for i=1:n_cor_groups
 eX(1,1)=rep_horiz(1,i);
 eX(2,1)=rep_horiz(2,i);
 rad=rep_radial(i,1);
 norm_rad_at_99(i,1)=rad/(3.035*rad*(eX'*cov_pred^-1*eX)^-0.5);
 if(norm_rad_at_99(i,1)<1)
     sum1=sum1+1;
 end
 norm_rad_at_90(i,1)=rad/(2.146*rad*(eX'*cov_pred^-1*eX)^-0.5);
 if(norm_rad_at_90(i,1)<1)
     sum2=sum2+1;
 end
 norm_rad_at_50(i,1)=rad/(1.177*rad*(eX'*cov_pred^-1*eX)^-0.5);
 if(norm_rad_at_50(i,1)>1)
     sum3=sum3+1;
 end
end
```

```
%norm_rad_at_99
%norm_rad_at_90
%norm_rad_at_50

pass_99_test=sum1/n_cor_groups
pass_90_test=sum2/n_cor_groups
pass_50_test=sum3/n_cor_groups

if(pass_99_test>=Y_h_99)
    pass_99='true'
end
if(pass_90_test>=Y_h_90)
    pass_90='true'
end
if(pass_50_test>=Y_h_50)
    pass_50='true'
end
```

G.1.4 Comparion of sub-method results of the "representative error samples" approach

```
% "TGD2c_hor_norm_representative_error_2_suboptions"    7/19/16

%This program investigates the effect on {probability of passing 90%
%probability-level test with a 90% confidence} of the two suboptions for
%the representative error approach for handling correlated samples
%as a function of the correlation of the n_cor_samples (specifiable) per
%original i.i.d samples, the latter assumed to equal 100, but specifiable.
%There are also 500 realizations, but specifiable as well.  This program
%is for horizontal errors.

%It also computes lub_h_90 (and best estimate be) accuracy based on
%order samples using the representententative i.i.d. errors vs. original i.i.d.
%errors, done for both suboptions and assumes 100 original iid samples for
%lub; true predicted (best estimate)CE90 equals (1.7371)*2m=3.47 since
%sqrt_ratio_eigen=0.5.  Actually average lub value and avg be value
%is computed over realization and results presented vs. correlation.


n_iid_samples=100

n_cor_samples=6  %number of cor samples per iid sample yielding
%a representative iid sample (samples part of "correlation sub-collection")

if(n_iid_samples~=100)
  warn='not 100 samples, lub incorrect'
end

n_cor_cases=6  %number of different correlation values cases

sample_sigma_bias_flag=0  %if 1, compute sample_sigma as biased estimate
%(computation involves divide by n instead of (n-1)) for suboption 1
```

```
n_real=500

cov_actual=eye(2);    % horizontal radials; a generic "actual covariance"
                      % common to each horizontal error sample;
                      % assumed same as predicted covariance cov
cov_actual(2,2)=4;    %generalize somewhat to different sigmas
%cov_actual(1,2)=1;    %make non-diagonal
%cov_actual(2,1)=1;    %make non-diagonal
cov=cov_actual;
cov_actual_sqrt=sqrtm(cov_actual);

cov_actual_group=eye(2*n_cor_samples); %covariance for all n_cor_sample
                 %corrrelated horizontal error, actual calues set in code

results_1=zeros(n_cor_cases,1);
results_2=zeros(n_cor_cases,1);

avg_acc_order_rep_1_lub=zeros(n_cor_cases,1);
avg_acc_order_rep_2_lub=zeros(n_cor_cases,1);
avg_acc_order_orig_lub=zeros(n_cor_cases,1);
avg_acc_order_rep_1_be=zeros(n_cor_cases,1);
avg_acc_order_rep_2_be=zeros(n_cor_cases,1);
avg_acc_order_orig_be=zeros(n_cor_cases,1);

    for j=1:n_cor_cases

        cor=(j-1)/(n_cor_cases-1);
        if(cor==1)
            cor=0.999;
        end

        for i1=1:n_cor_samples
            for j1=1:n_cor_samples
                for i2=1:2
                    for j2=1:2
                        k=(i1-1)*2+i2;
                        l=(j1-1)*2+j2;
                        cov_actual_group(k,l)=cov_actual(i2,j2);
                        if(i1~=j1)
                           cov_actual_group(k,l)=cor*cov_actual_group(k,l);
                        end
                    end
                end
            end
        end

        cov_actual_group_sqrt=sqrtm(cov_actual_group);

        temp_results_1=zeros(n_real,1);
        temp_results_2=zeros(n_real,1);
```

```
        for m=1:n_real

            sum_90_1=0;
            sum_90_2=0;

            samples_rep_1=zeros(n_iid_samples,1);
            samples_rep_2=zeros(n_iid_samples,1);
            samples_orig=zeros(n_iid_samples,1);

        for k=1:n_iid_samples

           s=cov_actual_group_sqrt*randn(2*n_cor_samples,1); %cor hor error
                                                             %samples
           s_temp=zeros(2,1);
           mean_X_samp=zeros(2,1);
           sigma_X_samp=zeros(2,2);
           tempX=zeros(2,1);

           %for accuracy computation
           samples_orig(k,1)=sqrt(s(1,1)^2+s(2,1)^2);


              for ij=1:n_cor_samples %compute sample stats over cor errors:

                  s_temp(1,1)=s((ij-1)*2+1,1);
                  s_temp(2,1)=s((ij-1)*2+2,1);

                  mean_X_samp(1,1)=mean_X_samp(1,1)+s_temp(1,1);
                  mean_X_samp(2,1)=mean_X_samp(2,1)+s_temp(2,1);

              end

              mean_X_samp=mean_X_samp/n_cor_samples;

              for ij=1:n_cor_samples %compute sample stats over cor errors

              sigma_X_samp=sigma_X_samp+(s_temp-mean_X_samp)*...
                           (s_temp-mean_X_samp)';

              end

              sigma_X_samp=sigma_X_samp/(n_cor_samples);  %suboption  1  biased
estimate
              if(sample_sigma_bias_flag~=1)
                  sigma_X_samp=sigma_X_samp*(n_cor_samples/(n_cor_samples-
1));
              end


              %Suboption 1:

              tempX(1,1)=sqrt(mean_X_samp(1,1)^2+...
                           sigma_X_samp(1,1));
```

```
            tempX(2,1)=sqrt(mean_X_samp(2,1)^2+...
                        sigma_X_samp(2,2));

            temp=sqrt(tempX'*(cov^-1)*tempX);  % cov assumed
                     %common and diagonal across samples, if not take avg
                     %cov and zero off-diagonals


            avg_90=temp/2.146;

            if((avg_90<1))  %chek if representative i.i.d sample passed
                sum_90_1=sum_90_1+1;
            end

            %representative samples for accuracy

            samples_rep_1(k,1)=sqrt(tempX(1,1)^2+tempX(2,1)^2);

            %Suboption 2:


          sigma_X_samp=(n_cor_samples/(n_cor_samples-1))*sigma_X_samp;

          if(sample_sigma_bias_flag~=1)%check suboption 1 bias flag
               sigma_X_samp=sigma_X_samp*((n_cor_samples-
1)/n_cor_samples);
            end

            tempX(1,1)=sqrt((abs(mean_X_samp(1,1))+0.5*...
                        sqrt(sigma_X_samp(1,1)))^2);
            tempX(2,1)=sqrt((abs(mean_X_samp(2,1))+0.5*...
                        sqrt(sigma_X_samp(2,2)))^2);

            temp=sqrt(tempX'*(cov^-1)*tempX);  %cov assumed
                     %common and diagonal across samples, if not take avg
                     %cov and zero off-diagonals


            avg_90=temp/2.146;

            if((avg_90<1))  %chek if representative i.i.d sample passed
                sum_90_2=sum_90_2+1;
            end

             %representative samples for accuracy

            samples_rep_2(k,1)=sqrt(tempX(1,1)^2+tempX(2,1)^2);

        end %end loop over i.i.d.samples

        temp_results_1(m,1)=sum_90_1/n_iid_samples;
        temp_results_2(m,1)=sum_90_2/n_iid_samples;
```

```
         samples_rep_1=sort(samples_rep_1);
         samples_rep_2=sort(samples_rep_2);
         samples_orig=sort(samples_orig);
         ii=95;%assumes 100 samples

        avg_acc_order_rep_1_lub(j,1)=
avg_acc_order_rep_1_lub(j,1)+samples_rep_1(ii ;
        avg_acc_order_rep_2_lub(j,1)=
avg_acc_order_rep_2_lub(j,1)+samples_rep_2(ii);

avg_acc_order_orig_lub(j,1)=avg_acc_order_orig_lub(j,1)+samples_orig(ii);

        ii=n_iid_samples*0.90;
        avg_acc_order_rep_1_be(j,1)=
avg_acc_order_rep_1_be(j,1)+samples_rep_1(ii);
        avg_acc_order_rep_2_be(j,1)=
avg_acc_order_rep_2_be(j,1)+samples_rep_2(ii);

avg_acc_order_orig_be(j,1)=avg_acc_order_orig_be(j,1)+samples_orig(ii);

        end % end loop over realizations

        avg_acc_order_rep_1_lub(j,1)= avg_acc_order_rep_1_lub(j,1)/n_real;
        avg_acc_order_rep_2_lub(j,1)= avg_acc_order_rep_2_lub(j,1)/n_real;
        avg_acc_order_orig_lub(j,1)=avg_acc_order_orig_lub(j,1)/n_real;
        avg_acc_order_rep_1_be(j,1)= avg_acc_order_rep_1_be(j,1)/n_real;
        avg_acc_order_rep_2_be(j,1)= avg_acc_order_rep_2_be(j,1)/n_real;
        avg_acc_order_orig_be(j,1)=avg_acc_order_orig_be(j,1)/n_real;

         ij=floor(0.9*n_real);
         temp_results=sort(temp_results_1,'descend');
         results_1(j,1)=temp_results(ij,1); %the percent of representative iid
          %samples passing the 90% probability normalized error test with
          %at least 90% condfidence for cor case j and suboption 1
         temp_results=sort(temp_results_2,'descend');
         results_2(j,1)=temp_results(ij,1); %suboption 2

    end   %end loop over cor cases

    avg_acc_order_rep_1_lub
    avg_acc_order_rep_2_lub
    avg_acc_order_orig_lub
    avg_acc_order_rep_1_be
    avg_acc_order_rep_2_be
    avg_acc_order_orig_be

figure (1)
clf  %this clears the figure from the previous run
hold on
plot1=100*results_1;
plot2=100*results_2;
cor_x=zeros(n_cor_cases,1);
```

```
 for j=1:n_cor_cases
        cor_x(j,1)=(j-1)*.2;
 end
plot(cor_x,plot2,'r',cor_x,plot1,'b','LineWidth',2);
%plot(cor_x,plot1,'r',cor_x,plot3,'g','LineWidth',2);
axis([0 1 50 100]);
title('Norm hor error test: suboption1(b), suboption2(r)')
xlabel('correaltion of n cor samples per iid sample')
ylabel('% realizations pass 90% probability normalized error test');
hold off
```

G.2 Options for sub-method 1 of the representative error samples approach

Predicted covariance matrix

The algorithm/equations for sub-method 1 of the "representative error samples" approach were presented as Equation (5.6.4.1-1) for horizontal error samples. It assumed the use of a common diagonal predicted error covariance for all of the error samples from the same sub-collection. If the corresponding predicted error covariance matrices are not common, simply use their average as an approximation. This approximation only affects the validation of predicted accuracy – it has no effect on the validation of accuracy as the predicted error covariance is not used. Since the error samples correspond to the same set of sensor data, the error covariance matrices are expected to be nearly the same in a given sub-collection; hence the effects of the approximation should be minimal. Furthermore, "diagonalize" the resultant predicted error covariance, i.e., ignore any resultant off-diagonal elements – simply use the resultant square-roots of the diagonal elements, σ_x and σ_y, per the algorithm.

(A potential issue concerns the above diagonalization if the corresponding predicted error covariance matrix eigenvalues are significantly different (ratio of the square-root of the smaller to larger eigenvalue approximately less than nominally 0.5) and the absolute value of the covariance matrix correlation coefficient between the x and y error components too large (nominally greater than 0.5). This is a potential area for future research.)

Algorithm/Equation (5.6.4.1-1) changes for vertical error samples

- Compute the sample mean and sample standard deviation about the sample mean for each error component over the k samples of ϵz: $mean\_z$, $sigma\_z$, (G-2)
    - both computation of the sample mean and sample standard deviation are unbiased estimates – see Section 5.2.1 of TGD 2b for the explicit calculations
- $ev = ((mean\_z)^2 + (sigma\_z)^2)^{\frac{1}{2}}$
- $ev\_norm = (((mean\_z)^2 + (sigma\_z)^2)/\sigma_z^2)^{1/2}$
- $\epsilon v\_norm_{99} = \epsilon v\_norm/2.576$, $\epsilon v\_norm_{90} = \epsilon v\_norm/1.645$, $\epsilon v\_norm_{50} = \epsilon v\_norm/0.674$
- test: $\epsilon v\_norm_{99} < 1$, $\epsilon v\_norm_{90} < 1$, and $\epsilon v\_norm_{50} > 1$
- $rep\_pred\_radial_{XX} = \epsilon v\_norm_{XX}/\ \epsilon v$, for plotting
- In addition, $\epsilon v\_norm_{XX} = ev/LE\_XX$, where $XX = 99{,}90{,}50$, if scalar accuracy metrics are to be used

Proceed with baseline accuracy and predicted accuracy validation for vertical errors using the resultant vertical radial error ev and the normalized vertical radial errors at the three different probability levels that correspond to the representative vertical error sample from each of the correlated sub-collections.

Algorithm/Equation (5.6.4.1-1) changes for 3d error samples

- Compute the sample mean and sample standard deviation about the sample mean for each error component over the k samples of $\epsilon X = [\epsilon x \quad \epsilon y \quad \epsilon z]^T$: $mean\_x, sigma\_x, mean\_y, sigma\_y, mean\_z, sigma\_z$ (G-3)
  - both computation of the sample mean and sample standard deviation are unbiased estimates – see Section 5.2.1 of TGD 2b for the explicit calculations
- $er = ((mean\_x)^2 + (sigma\_x)^2 + (mean\_y)^2 + (sigma\_y)^2 + (mean\_z)^2 + (sigma\_z)^2)^{\frac{1}{2}}$
- $er\_norm = \left(\frac{(mean\_x)^2 + (sigma_x)^2)}{\sigma_x^2} + \frac{\left(mean\_y)^2 + (sigma_y\right)^2)}{\sigma_y^2} + \frac{(mean\_z)^2 + (sigma_z)^2)}{\sigma_z^2} \right)^{1/2}$
- $\epsilon r\_norm_{99} = \epsilon r\_norm/3.368$, $\epsilon r\_norm_{90} = \epsilon r\_norm/2.500$, $\epsilon r\_norm_{50} = \epsilon r\_norm/1.538$
- test: $\epsilon r\_norm_{99} < 1$, $\epsilon r\_norm_{90} < 1$, and $\epsilon r\_norm_{50} > 1$
- $rep\_pred\_radial_{XX} = \epsilon r\_norm_{XX}/\ \epsilon r$, for plotting
- In addition, $\epsilon r\_norm_{XX} = er/SE\_XX$, where $XX = 99,90,50$, if scalar accuracy metrics are to be used

Proceed with baseline accuracy and predicted accuracy validation for 3d errors using the resultant 3d radial error er and the normalized 3d radial errors at the three different probability levels that correspond to the representative 3d error sample from each of the correlated sub-collections.

Appendix H: Recommended Number of i.i.d. Error Samples including Pseudo-code

This appendix supports Section 4.1 of the TGD 2c which contains an overview of the specification and validation of accuracy, and Section 4.1.1 in particular. It presents additional results and documents the (non-optimized) MATLAB pseudo-code which the results were based on.

Horizontal errors were simulated based on a mean-zero multi-variate Gaussian probability distribution; hence, results are approximate as this is an assumption. Order statistics do not require an assumption regarding the underlying probability distribution, but for results based on Monte-Carlo simulation, a distribution must be specified.

H.1 Validation pad versus number of error samples

Simulation results correspond to the expected value of the 90% confidence-level or validation "pad" relative to the true 90th percentile of horizontal errors (aka "CE_true" in the code below). Results also include the 90th percentile of the above pad, termed the max_rel_pad. Results are presented as a function of the assumed number of i.i.d. error samples used for the validation of accuracy: 25, 40, 55, 100, 150, 200, 400.

(Note: 50 was not included explicitly as a number of i.i.d.samples – it corresponds to a relatively small anomalous spike, i.e., the corresponding ordered sample # is 49 as it is the smallest ordered sample such that the probability of the lub exceeding the true 90th percentile is greater than 90%, in this case 95%; if ordered sample # 48 were selected instead, the corresponding probability would only be 88%. Virtually all of the ordered samples selected for the other number of i.i.d. samples correspond to a probability between 91 – 93% (see TGD 2b, Table 5.3.3.2-2). Also, this phenomena is inconsequential (probability not much bigger than 91%) if a relatively large number of samples.

Consequently, interpolating max_rel_pad in the main body of this document between 40 and 55 samples for 50 samples, we get about 35% and with that design margin a probability of only 80% of correctly passing validation. If we allow a spike for max_rel_pad at an explicit number of samples equal to 50, we get a required design margin of about 42% to get 90% correctly passing validation using ordered sample #49.

The above is relatively unimportant as we are looking for general principles regarding the recommended number of samples.

The only alternative to this type of phenomena is to redefine the value of the lub as an interpolation of the value of the first ordered sample that exceeds 90% with the value of the previous ordered sample based on the difference of the two corresponding probabilities and such that the result corresponds to an approximate 90% probability. This is a possible area for future research.

Finally, as a reminder, for a given number of i.i.d. samples, the same ordered sample # is selected for each realization of the simulation, but has a different (lub) value associated with it.)

Prior to presenting the code, additional figures are first presented in order to supplement the results presented in Figure 4.1.1-2 of Section 4.1.1. The latter assumed a reasonable value for "ratio", or more precisely, the "sqrt_eigenvalue_ratio", equal to 0.8. Ratio is defined as the square-root of the minimum eigenvalue divided by the maximum eigenvalue of the corresponding true error covariance matrix – the smaller the value of ratio, the more elongated the corresponding error ellipse. For example, a ratio equal to 0.50 corresponds to an error ellipse with a semi-major axis twice as long as its semi-minor axis (see Section 5.4.2 of TGD 2a). Results are somewhat sensitive to the value of ratio. Figures H-1 through H-4 below assume values for ratio equal to 1.0, 0.8, 0.5, and 0.25, respectively. A ratio of 0.8 (and a slightly different scale for the y-axis) were used in Figure 4.1.1-2 in the main body of this document.

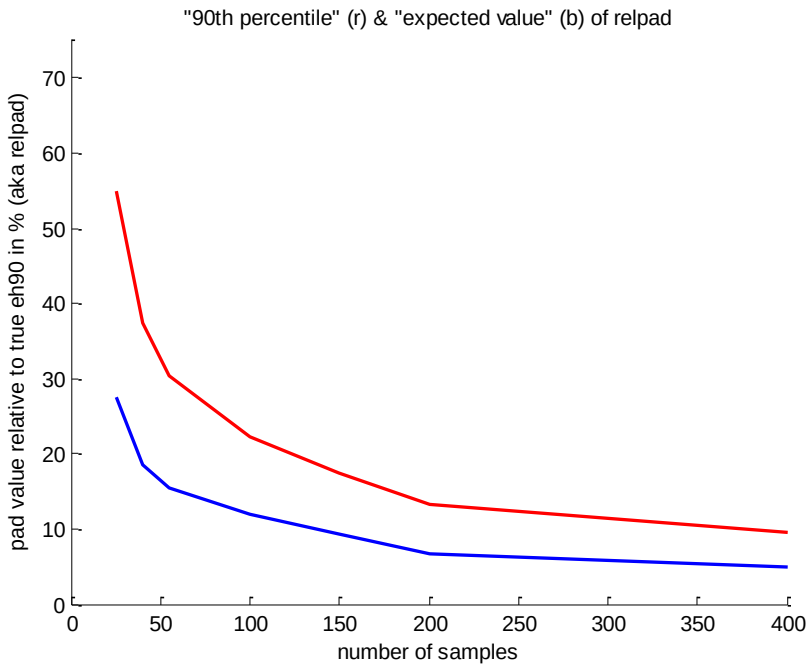


Figure H-1: Expected value (blue) and $ZZ = 90^{\text{th}}$ percentile (red) of the validation pad relative to the true value ϵh_{90} (%); ratio = 1.0.

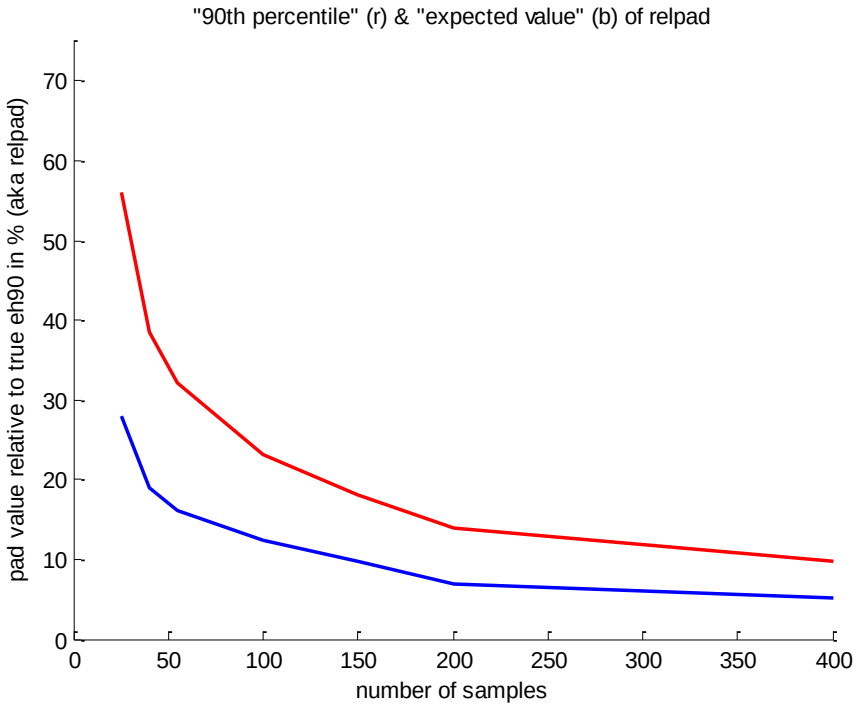


Figure H-2: Expected value (blue) and $ZZ = 90^{\text{th}}$ percentile (red) of the validation pad relative to the true value ϵh_{90} (%); ratio = 0.80.

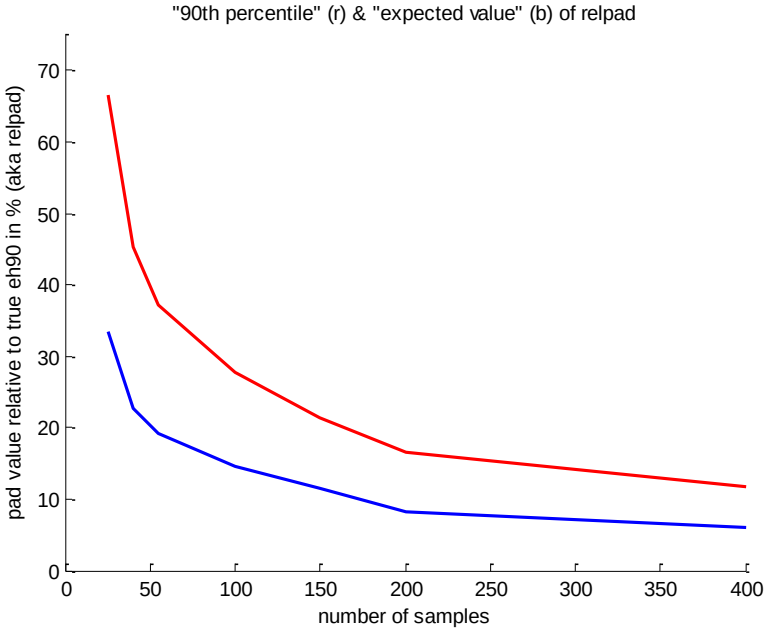


Figure H-3: Expected value (blue) and $ZZ = 90^{\text{th}}$ percentile (red) of the validation pad relative to the true value ϵh_{90} (%); ratio = 0.50.

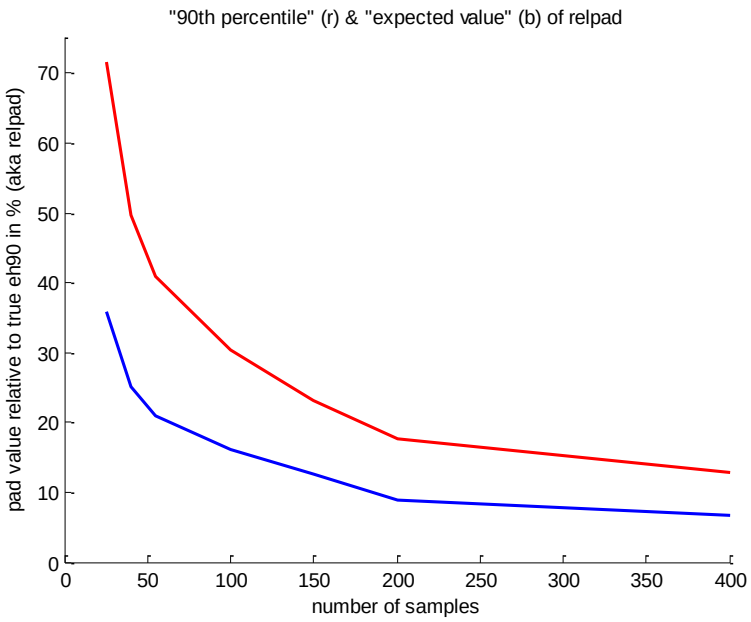


Figure H-4: Expected value (blue) and $ZZ = 90^{\text{th}}$ percentile (red) of the validation pad relative to the true value ϵh_{90} (%); ratio = 0.25.

Validation corresponds to horizontal accuracy, a $XX = 90^{\text{th}}$ percentile of horizontal radial error (ϵh_{90}), a $YY = 90\%$ confidence-level for the lub, and a $ZZ = 90^{\text{th}}$ percentile for the relative validation pad. However, the following pseudo-code can be modified in a straight-forward manner to accommodate any values of either XX or YY equal to 50, 90, or 95%, any positive value for ZZ less than 100%, as well as accommodate the validation of either vertical, horizontal, or 3d accuracy.

Pseudo-code:

```
% 10/01/16    "TGD2c_number_of_samples_investigation_3_more"

%compute the expected (average)value and the ZZth percentile of the 90%
%confidence-level or validation pad relative to the (true) eh_90 (aka
%rel_pad)

%the ratio of the sqrt of true eigenvalues does make somewhat of a
%difference - the computed pad increases with ratio decrease

ZZ=0.90

ratio_case_number=2       %select default "shape" of true error covariance
                          %matrix; value of 2 most representative for
                          %ratio = square root of ratio of min-to-max
                          %eigenvalues
n_sample_cases=7

sample_cases=[25,40,55,100,150,200,400]
lub_90_cases=[25,39,53,95,141,186,369] %one-sided conf interval(order stats)
```

```
n_real=5000

cov=eye(2);

if(ratio_case_number==1)
ratio=1;
CE90_true=2.146*1;  %ratio=1
end
if(ratio_case_number==2)
ratio=0.8;
cov(2,2)=1.5625;%ratio=0.8
CE90_true=1.9472*1.25;%ratio=0.25
end
if(ratio_case_number==3)
ratio=0.5;
cov(2,2)=4;%ratio=0.5
CE90_true=1.7371*2;%ratio=0.5
end
if(ratio_case_number==4)
ratio=0.25;
cov(2,2)=16;%ratio=0.25
CE90_true=1.6646*4;%ratio=0.25
end

ratio
cov
CE90_true %aka true eh90

expected_pad=zeros(n_sample_cases,1);
expected_pad_2=zeros(n_sample_cases,1);
max_pad=zeros(n_sample_cases,1);%ZZth percentile of rel_pad

for i=1:n_sample_cases    %number of samples value

    n_samples=sample_cases(1,i);
    lub_90_order=lub_90_cases(1,i);

    pad_dist=zeros(n_real,1);
    pad_sum=0;
    pad_sum_2=0;

    for j=1:n_real     %loop over realizations

        h_radial=zeros(n_samples,1);

        for k=1:n_samples     %loop over samples

            X=cov^0.5*randn(2,1);
            h_radial(k,1)=sqrt(X(1,1)^2+X(2,1)^2);

        end %samples loop

        h_radial=sort(h_radial);
        lub_90=h_radial(lub_90_order,1);
        best_est_90=h_radial(floor(0.5+0.90*n_samples),1);
```

```
        diff=(lub_90-CE90_true)/CE90_true;
        diff_be=(lub_90-best_est_90)/best_est_90;

            pad_sum=pad_sum+diff;
            pad_sum_2=pad_sum_2+diff_be;
            pad_dist(j,1)=diff;

    end %real_loop

    avg_pad=pad_sum/n_real;
    avg_pad_2=pad_sum_2/n_real;

    expected_pad(i,1)=100*avg_pad;
    expected_pad_2(i,1)=100*avg_pad_2;

    pad_dist=sort(pad_dist);
    ik=floor(ZZ*n_real);
    max_pad(i,1)=100*pad_dist(ik,1);


end % sample cases loop

expected_pad  %expected value of rel_pad
expected_pad_2 %expected value of pad relative to best estimate of CE_90_true
max_pad  %ZZ percentile of rel_pad

max_x=sample_cases(n_sample_cases);
n_samp=zeros(n_sample_cases,1);
 for j=1:n_sample_cases
       n_samp(j,1)=sample_cases(1,j);
 end

figure (1)
clf  %this clears the figure from the previous run
hold on

plot(n_samp,max_pad,'r',n_samp,expected_pad,'b','LineWidth',2);

axis([0 max_x 0 75]);
title('"90th percentile" (r) & "expected value" (b) of relpad' )
xlabel('number of samples')
ylabel('pad value relative to true eh90 in % (aka relpad)');
hold off
```

H.2 Design Margin

In addition, Section 4.1.1 illustrated the ability to determine the probability of successful validation as a function of the number of i.i.d. samples and an assumed value for "design margin". Validation is assumed to correspond to horizontal accuracy, and an $XX = 90\%$ percentile and $YY = 90\%$ confidence-level. The value for "ratio" can be selected as well.

The following output is from the first set of pseudo-code that follows:

>> TGD2c_number_of_samples_investigation_design_margin

n_sample_case_number = 2

design_margin =0.3000

ratio_case_number 2

sample_cases =25 50 100 150 200 400 600

lub_90_order_cases =25 48 95 141 186 369 550

n_samples =50

lub_90_order = 48

n_real =5000

ratio = 0.8000

cov = 1.0000 0

0 1.5625

CE90_true = 2.4340

CE90_spec =3.1642

prob_success_validation =0.9066

Pseudeo-code:

```
% "TGD2c_number_of_samples_investigation_design margin"    9/16/16

%compute probability of passing validation (via Monte Carlo simulation) for
%assumed XX=90% horizontal radial error percentile and
%YY=90 confidence level used in the validation of accuracy; assume a
%specific design margin

%(Note: following algorithm/code easily modified to accomodate
%specified XX=50, 90, or 95; specified YY= 50, 90, or 95;
%specified vertical, horizontal, or 3d radial errors)

%validation based on comparing lub at 90% confidence-level computed
%using order stats to a CE90spec equal to the
%true CE90 plus the design margin

%specify the number of i.i.d. samples

%specify the assumed design margin relative to the true value of the 90th
%percentile of horizontal error

%specify the assumed representative shape ("ratio") of the error covariance
%matrix of ellipse, and corresponding true CE90


n_sample_case_number=2    %select the number of samples from "sample cases"
                          %below

design_margin=0.30       %specify assumed design margin relative to true value
                         %of 90% horizontal radial error percentile

ratio_case_number=2       %select default "shape" of true error covariance
                          %matrix; value of 2 most representative for
                          %ratio = square root of ratio of min-to-max
                          %eigenvalues

sample_cases=[25,50,100,150,200,400,600]
%lub_90_order)cases=[25,48,95,141,187,370,552] %two-sided conf interval
lub_90_order_cases=[25,48,95,141,186,369,550] %one-sided conf interval

n_samples=sample_cases(n_sample_case_number)
lub_90_order=lub_90_order_cases(n_sample_case_number)

n_real=5000

cov=eye(2);    %2x2 corresponding to horizontal errors; overall common
               %scaling will not affect output

ratio=1;
cov=eye(2); %ratio=1
CE90_true=2.146*1;  %ratio=1
if(ratio_case_number==2)
```

```
ratio=0.8;
cov(2,2)=1.5625;%ratio=0.8
CE90_true=1.9472*1.25;%ratio=0.25
end
if(ratio_case_number==3)
ratio=0.5;
cov(2,2)=4;%ratio=0.5
CE90_true=1.7371*2;%ratio=0.5
end
if(ratio_case_number==4)
ratio=0.25;
cov(2,2)=16;%ratio=0.25
CE90_true=1.6646*4;%ratio=0.25
end

ratio
cov
CE90_true
CE90_spec=(1+design_margin)*CE90_true

    count=0;

    for j=1:n_real

        h_radial=zeros(n_samples,1);

        for k=1:n_samples

            X=cov^0.5*randn(2,1);
            h_radial(k,1)=sqrt(X(1,1)^2+X(2,1)^2);

        end %samples loop

        h_radial=sort(h_radial);
        lub_90=h_radial(lub_90_order,1);

       if(lub_90<CE90_spec)
            count=count+1;
        end

    end %real_loop

   prob_success_validation=count/n_real
```

Probability of Validaton passing, Type I errors, and Type II errors

The following pseudo-code supports the generation of Figures 4.1.1-3 through 4.1.1-5:

```
% 10/05/16    "TGD2c_number_of_samples_investigation_type_errors"

%Monte Carlo simulation of probability of passing validation vs amount of
%design margin; lub confidence level Y=90%.

%By definition, CE90_spec=(1+design_margon)*eh_90, where eh_90 is the true
%horizontal radial error (true CE90)

%If validation passes and design margin less that true CE90: a Type II
errors;
%if validation does not pass and design margin greater than true CE90: a Type
%I error.

n_sample_cases=5
sample_cases=[25,40,55,100,200]
lub_90_cases=[25,39,53,95,186] %one-sided conf interval(order stats)

n_test_cases=17
test_min=-0.20
test_step=0.05

n_real=4000

ratio_case_number=2      %select default "shape" of true error covariance
                         %matrix; value of 2 most representative for
                         %ratio = square root of ratio of min-to-max
%eigenvalues
cov=eye(2);
if(ratio_case_number==1)
ratio=1;
CE90_true=2.146*1;  %ratio=1
end
if(ratio_case_number==2)
ratio=0.8;
cov(2,2)=1.5625;%ratio=0.8
CE90_true=1.9472*1.25;%ratio=0.25
end
if(ratio_case_number==3)
ratio=0.5;
cov(2,2)=4;%ratio=0.5
CE90_true=1.7371*2;%ratio=0.5
end
if(ratio_case_number==4)
ratio=0.25;
cov(2,2)=16;%ratio=0.25
CE90_true=1.6646*4;%ratio=0.25
end
ratio
cov
CE90_true %aka true eh90
```

```
test=zeros(n_sample_cases,n_test_cases);

for i=1:n_sample_cases    %number of samples value

    n_samples=sample_cases(1,i);
    lub_90_order=lub_90_cases(1,i);


    for j=1:n_real     %loop over realizations

        h_radial=zeros(n_samples,1);

        for k=1:n_samples     %loop over samples

            X=cov^0.5*randn(2,1);
            h_radial(k,1)=sqrt(X(1,1)^2+X(2,1)^2);

        end %samples loop

        h_radial=sort(h_radial);
        lub_90=h_radial(lub_90_order,1);

        for k=1:n_test_cases     %loop over test cases

            temp=test_min+(k-1)*test_step;
            spec=(1+temp)*CE90_true;

            if(lub_90<spec)
                test(i,k)= test(i,k)+1;
            end

        end    %test  cases  loop

    end    %real   loop


end   %samples loop

test=100*test/n_real

    rel_spec=zeros(n_test_cases,1);
    test_25=zeros(n_test_cases,1);
    test_40=zeros(n_test_cases,1);
    test_55=zeros(n_test_cases,1);
    test_100=zeros(n_test_cases,1);
    test_200=zeros(n_test_cases,1);

    for i=1:n_test_cases
        rel_spec(i,1)=100*(test_min+(i-1)*test_step);
    end
```

```
        for j=1:n_test_cases
            test_25(j,1)=test(1,j);
            test_40(j,1)=test(2,j);
            test_55(j,1)=test(3,j);
            test_100(j,1)=test(4,j);
            test_200(j,1)=test(5,j);
        end

        figure (1)
        clf  %this clears the figure from the previous run
        hold on
        plot(rel_spec,test_25,'r',rel_spec,test_40,'g',...
            rel_spec,test_55,'c',rel_spec,test_100,'b',...
            rel_spec,test_200,'m','LineWidth',3);
        axis([-20 60 0 100]);
        title('prob val. pass vs. design margin; nmbr samples: 25,r; 40,g;
55,cy; b,100; m,200' )
        xlabel('design magin (percent)')
        ylabel('prob of validation passing (percent)');
        hold off

        figure (2)
        clf  %this clears the figure from the previous run
        hold on
        plot(rel_spec,test_25,'r',rel_spec,test_40,'g',...
            rel_spec,test_55,'cy',rel_spec,test_100,'b',...
            rel_spec,test_200,'m','LineWidth',3);
        axis([-20 0 0 100]);
        title('prob of Type II error vs. design margin; nmbr samples: 25,r;
40,g; 55,cy; b,100; m,200' )
        xlabel('design margin (percent)')
        ylabel('prob of validation passing (percent)');
        hold off


        for j=1:n_test_cases
            test_25(j,1)=100-test(1,j);
            test_40(j,1)=100-test(2,j);
            test_55(j,1)=100-test(3,j);
            test_100(j,1)=100-test(4,j);
            test_200(j,1)=100-test(5,j);
        end

        figure (3)
        clf  %this clears the figure from the previous run
        hold on
        plot(rel_spec,test_25,'r',rel_spec,test_40,'g',...
            rel_spec,test_55,'cy',rel_spec,test_100,'b',...
            rel_spec,test_200,'m','LineWidth',3);
        axis([0 60 0 100]);
        title('prob of Type I error vs. design margin; nmbr samples: 25,r;
40,g; 55,cy; b,100; m,200' )
        xlabel('design margin (percent)')
        ylabel('prob of validation not passing (percent)');
        hold off
```

Appendix I: Conversions for the values of Specified Accuracy

Section I.1 of this appendix addresses the conversion of root-mean-square error to scalar accuracy metrics. Section I.2 addresses the conversion of scalar accuracy metrics from one probability level to another. Both types of conversion are for specified accuracy requirements only (e.g., $CE90_{spec}$).

I.1 Conversion of rmse to scalar accuracy metrics

Some sensor systems, including some commercial satellite imaging systems, specify/advertise their accuracy in terms of root-mean-square error (rmse). Its corresponding value was typically derived or "validated" through the use of sample statistics.

In particular, for horizontal radial errors, rmse is defined as follows:

$$rmse \equiv \sqrt{E\{\epsilon x^2 + \epsilon y^2\}} = \sqrt{E\{\epsilon h^2\}} \neq E\{\sqrt{\epsilon h^2}\}\,, \tag{I-1}$$

Where the random variable $\epsilon h \equiv \sqrt{\epsilon x^2 + \epsilon y^2}$ and $E\{\}$ is the expected value operator.

The sample rmse is computed as follows:

$$rmse_s = \sqrt{\frac{\sum_{i=1}^{n} \epsilon h_i^2}{n}}, \tag{I-2}$$

where samples $\epsilon h_i = \sqrt{\epsilon x_i^2 + \epsilon y_i^2}$ and n is the total number of independent samples.

For an accuracy specification based on $rmse$, validation would simply consist of computing $rmse_s$ and checking that it is less than or equal to an $rmse_{spec}$, the latter set to the specified/advertised accuracy, possibly with an additional margin included. This is not adequate for NSG purposes because a corresponding probability is not supplied as well as other limitations discussed below.

In order to convert (sample) rmse to a corresponding probability-level, a probability distribution must be assumed, typically an independent mean-zero Gaussian distribution for both ϵx and ϵy with a common standard deviation σ, and therefore, a Rayleigh distribution for the random variable ϵh. If so, and assuming a reasonable number of samples, $rmse_s$ corresponds to a value approximately equal to $rmse = \sqrt{2}\sigma$ with a corresponding probability of only 63% that an arbitrary horizontal radial error is less than or equal to it. (This follows from evaluation of the Rayleigh cumulative distribution function for ϵh which is equal to $cdf(\epsilon h) = 1 - e^{\frac{-eh^2}{2\sigma^2}}$, and which, when evaluated at $eh = rmse = \sqrt{\sigma^2 + \sigma^2} = \sqrt{2}\sigma$, equals 0.632.)

The above assumption regarding a probability distribution was required in order to generate a specific probability-level that corresponds to rmse. It is a significant restriction and non-robust as compared to the recommended use of horizontal radial error XX percentiles (or equivalently, $CEXX$) for the specification of accuracy, where XX (50, 90, or 95%) corresponds directly to the probability-level. A

corresponding best estimate and least-upper-bound of the horizontal radial error XX percentile are computed using order statistics for the validation of accuracy. Order statistics are independent of probability distribution (including a possible non-zero mean-value).

Given geolocation accuracy that is specified/advertised in terms of rmse for a sensor, the best alternative for an NSG geolocation system that is to integrate this sensor into a geolocation capability is to first convert the rmse into an approximate scalar accuracy metric $CEXX$ equivalent, and then proceed with the baseline method for the specification of accuracy and its validation using order statistics as documented in the main body of the document.

Unfortunately, the above conversion must assume a specific probability distribution for underlying errors – this is unavoidable. However, the good news is that after conversion, order statistics can be used for validation of the accuracy requirement. In other words, validation based on sample rmse is avoided, as desired. Validation based on order statistics is used instead, and can "detect" the effects of most larger-valued (and lower probability) horizontal radial errors, whereas validation based on sample rmse tends to mask their effect.

We expand the conversion procedure discussed earlier corresponding to a Rayleigh distribution for the horizontal radial error random variable ϵh in order to generate $CEXX$, for XX = 50, 90, and 95%, which are equivalent to rmse. That is, the underlying horizontal error $\epsilon X = [\epsilon x \quad \epsilon y]^T$ is still assumed to have an independent mean-zero Gaussian probability distribution with a common standard deviation σ for each error component or random variable ϵx and ϵy.

The easiest way to perform this expanded conversion is to use Table 5.4.2-1 of TGD 2a (predictive statistics), and observe that both eigenvalues of the corresponding $2x2$ covariance matrix for horizontal error $\epsilon X = [\epsilon x \quad \epsilon y]^T$ are equal to σ^2, and thus the square-root of the maximum eigenvalue equals σ and the ratio of the square-root of the eigenvalues equals 1. The table states that $CE50 = 1.1174\sigma$, $CE90 = 2.1460\sigma$, and $CE95 = 2.4478\sigma$. And substituting $rmse/\sqrt{2}$ for σ, we obtain the appropriate conversion factors of Table I.1-1.

Thus, for example:

$$CE90 = 1.5174\, rmse. \qquad \text{(A-3)}$$

Table I.1-1: RMSE to Scalar Accuracy Metrics conversion factors

	XX = 50%	XX = 90%	XX = 95%
LE_XX	0.6745	1.6499	1.9600
CE_XX	0.8325	1.5175	1.7309
SE_XX	0.8881	1.4435	1.6140

Note that Table I.1-1 also presents conversion factors for rmse of vertical (radial) errors to $LEXX$ and for rmse of (3d) spherical radial errors to $SEXX$. For vertical radial errors, $rmse \equiv \sqrt{\epsilon z^2}$, and for (3d) spherical radial errors, $rmse \equiv \sqrt{\epsilon x^2 + \epsilon y^2 + \epsilon z^2}$.

The conversion factors in Table I.1-1 for $LEXX$ can be read directly from Table 5.4.1-1 of TGD 2a, and the conversion factors for $SEXX$ can be read directly from Tables 5.4.3-1,2,3 of TGD 2a using the same procedure as for $CEXX$, but also accounting for the square-root of the eigenvalues equal to $rmse/\sqrt{3}$ instead of $rmse/\sqrt{2}$.

One last comment regarding rmse conversions in general, using horizontal radial error as an example: It is conceivable that both the rmse of x-component error and the rmse of y-component error are provided instead of the rmse of horizontal radial error. If so, simply equate the latter with the root-sum-square of the former, i.e., $rmse\ \epsilon h = \sqrt{(rmse\ \epsilon x)^2 + (rmse\ \epsilon y)^2}$. Furthermore, in the unlikely event that instead of the rmse of x-component error, its non-zero sample mean μ_{x_s} and its sample standard deviation σ_{x_s} are provided instead, first compute $rmse\ \epsilon x = \sqrt{\mu_{x\_s}^2 + \sigma_{x\_s}^2}$. (A similar computation is applicable to the y-component of error as well.)

I.2 Conversion of scalar accuracy metrics from one probability level to another

It is possible that a scalar accuracy metric is to be converted from one probability level to another for convenience in order to supply a scalar accuracy metric for the specification of accuracy. The following conversion factors (Table I.2-1) are approximate in that they assume an underlying mean-zero Gaussian probability distribution for the underlying error components (random variables) ϵx, ϵy, and/or ϵz which are independent (uncorrelated) and with equal variances (standard deviations). They are based on the partial results of Section 5.4 of TGD 2a and correspond to predictive statistics.

Table I.2-1: Conversion of Scalar Accuracy Metrics from one probability level to another

FROM:	TO:								
	LE_50	LE_90	LE_95	CE_50	CE_90	CE_95	SE_50	SE_90	SE_95
LE_50	1	2.4461	2.9059						
LE_90	0.4088	1	1.1880						
LE_95	0.3441	0.8418	1						
CE_50				1	1.8227	2.0790			
CE_90				0.5486	1	1.1406			
CE_95				0.481	0.8767	1			
SE_50							1	1.6255	1.8174
SE_90							0.6152	1	1.1181
SE_95							0.5502	0.8944	1

Thus, for example, using Table I.2-1:

LE_95 = 1.1880 x LE_90, and

CE_90 = 0.8767 x CE_95.

Conversions are to be applied for the specification of accuracy, not for the validation of specified accuracy. That is, they are not to be applied to the best estimate or least-upper-bound of radial error percentile XX generated using order statistics to a best estimate or least-upper-bound of radial error percentile YY.

For example, although ϵh_{XX}, the XX percentile of horizontal (radial) error, is defined equivalent to $CEXX$, the scale factors associated with $CEXX$ to $CEYY$ (Table I.2-1) are not applicable to their radial error percentile counterparts generated using order statistics. For example, assuming 100 samples, the best estimate of the 50th percentile of horizontal (radial) error is equal to the value of the 50th ordered sample, and independent of the actual values of the 51st to 100th ordered samples. The latter contain the 90th ordered sample which is equal to the best estimate of the 90th percentile of horizontal (radial) error– the two best estimates do not scale.